

ASSEMBLEUR ET PÉRIPHÉRIQUES DES MSX

PIERRE BRANDEIS
ET FRÉDÉRIC BLANC

PIERRE BRANDEIS
ET FRÉDÉRIC BLANC

ASSEMBLEUR ET PÉRIPHÉRIQUES DES MSX



ÉDITIONS DU P.S.I.
1985

Sommaire

Introduction

Vous possédez un ordinateur au standard MSX. Sans doute avez-vous composé sur celui-ci de nombreux programmes en BASIC. Et malgré les excellentes performances de l'appareil, peut-être n'avez-vous pu réaliser ce que vous projetiez, car le BASIC est par nature un langage assez « lent ». Faire de l'animation, un jeu d'action rapide ou un traitement de texte demande de très nombreuses opérations, beaucoup de tests, etc. On arrive vite à la limite de ce que peut donner le BASIC malgré l'aide précieuse des sprites pour les graphismes.

Pour aller plus loin, il devient nécessaire de programmer directement en langage machine.

Afin de comprendre cette nécessité, il faut revenir sur la structure du BASIC.

Le BASIC est un langage interprété, c'est-à-dire que pour exécuter un ordre tel que PRINT, par exemple, la machine doit isoler le mot-clé, puis le comparer à tous les ordres qu'elle peut comprendre pour le reconnaître, avant de pouvoir exécuter la routine en code machine qui lui correspond en ROM. Si l'on fait une boucle du type :

```
10 FOR I = 1 TO 100
20 PRINT PEEK (I)
30 NEXT I
```

le BASIC devra interpréter 100 fois chaque ordre (PRINT,PEEK (,),NEXT). Il devra rechercher 100 fois I dans la table des variables pour trouver l'argument de PEEK, puis de nouveau 100 fois pour l'incrémenter après avoir interprété NEXT. C'est un gros travail qui prend forcément du temps car l'interprétation d'un ordre demande de très nombreuses opérations élémentaires.

Cependant, l'avantage du BASIC est qu'il suffit de donner un ordre simple, clair, en anglais, pour obtenir un résultat. On peut très bien ignorer ce qui se passe réellement dans la machine, cela n'empêche pas de programmer.

Mais plus on se rapproche du langage de l'homme, plus on s'éloigne de celui de la machine. On crée donc un travail supplémentaire de transcription, et cela se paie en temps.

A l'inverse, moins le langage qu'on emploie sera évolué, moins chaque instruction demandera d'opérations, et plus rapide sera l'exécution.

Si l'on cherche la rapidité, la solution est de programmer en code machine : une tâche effectuée en code machine est de 10 à 100 fois plus rapide qu'une même tâche écrite en BASIC ! Cependant, le code machine est vraiment très abstrait, illisible, et éloigné du langage humain (car il n'est composé que de nombres binaires, suite de 0 et de 1) ; on ne peut envisager de l'ecrire directement. On va donc employer un moyen terme : on écrira dans un langage symbolique qu'un programme spécial se chargera de traduire en code machine. Ce programme s'appelle ASSEMBLEUR. On nomme langage machine ou par extension ASSEMBLEUR le langage symbolique dans lequel on écrit.

Outre la rapidité, vous gagnerez également en place mémoire ; un programme BASIC et son interpréteur prendront toujours beaucoup plus de place qu'un programme en langage machine réalisant la même chose.

Rien n'empêche d'ailleurs de mêler les langages : un squelette en BASIC appellera des sous-programmes en langage machine, là où la rapidité sera nécessaire.

Vous avez sans doute entendu dire que le langage machine est très compliqué, ardu, rébarbatif, difficile... En fait il n'en est rien, et si vous suivez les chapitres de ce livre, si vous faites les exercices proposés, vous ne devriez pas avoir trop de mal à développer vos programmes en ASSEMBLEUR, pardon, en langage machine !

Cet ouvrage est organisé en deux parties : la première est consacrée au langage machine du Z80 (microprocesseur des MSX) ; elle vous permettra d'apprendre les nombreuses instructions et de construire quelques petits programmes généraux.

La deuxième partie concerne les périphériques du MSX ainsi que les processeurs annexes. Vous y trouverez tous les renseignements concernant l'écran, le clavier, les joysticks et le générateur sonore. Elle vous permettra d'élaborer de nombreux programmes utilisant ces périphériques.

De plus, un chapitre est consacré à la mémoire morte des MSX, outil indispensable pour l'élaboration rapide de logiciels.

Sans plus attendre, passons à la première partie consacrée à l'ASSEMBLEUR du Z80.

ASSEMBLEUR DU Z80 | 1

1

Principes de base

NUMÉRATION POSITIONNELLE; BASES 10, 2, 16

L'ordinateur est bête ! Même très bête : il ne comprend rien, si ce n'est la présence ou l'absence de courant en certains de ses points. C'est pourquoi il ne peut pas comprendre des chiffres comme 5, 9, 45 ou 100, ni une lettre ou un signe comme « A », « + », ou « ? », etc.

Comment faire dans ce cas ? En multipliant le nombre de points où il peut y avoir ou non du courant, on arrive à travers les différentes combinaisons à coder de nombreuses informations.

Dans notre système décimal (c'est-à-dire en base 10), considérons le nombre 2034. Il s'énonce « deux mille trente quatre », mais on peut aussi l'écrire :

$$2034 = 2 \times 1000 + 0 \times 100 + 3 \times 10 + 4 \times 1$$

soit :

$$2034 = 2 \times 10^3 + 0 \times 10^2 + 3 \times 10^1 + 4 \times 10^0.$$

(Un nombre à la puissance 0 vaut 1 par définition quel que soit ce nombre).

On voit que chaque chiffre prend suivant la place qu'il occupe dans le nombre une valeur différente : c'est la numération de position. Le chiffre le plus à droite a le « poids » 1 (unités), le deuxième a le « poids » 10 (dizaines), le troisième le poids 100 (centaines), etc. Il existe en base 10 dix chiffres (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), le dernier valant neuf, soit (base −1).

Toutes les bases fonctionnent de manière similaire.

Dans la machine, il n'y a que deux états possibles : le courant passe ou ne passe pas. Appelons par définition état 0 le fait que le courant ne passe pas, et état 1 le fait que le courant passe en un point particulier. Ce point peut donc prendre deux valeurs, 0 ou 1 : nous sommes en base 2 ou **binaire.** En base 2, on aura deux chiffres, 0 et 1 ; 2 s'écrit 10 ($1 \times 2^1 + 0 \times 2^0$), 3 s'écrit 11, 4 s'écrit 100, etc.

Voici une table de conversion base 2-base 10 (c'est-à-dire binaire-décimal) :

0	=	0
1	=	1
10	=	2
11	=	3
100	=	4
101	=	5
110	=	6
111	=	7
1000	=	8
1001	=	9
1010	=	10
1011	=	11
1100	=	12
1101	=	13
1110	=	14
1111	=	15

En accolant huit points semblables, on obtient un **octet** de la forme :

11010111, par exemple.

Chaque chiffre binaire est appelé **bit** et a pour poids la puissance de 2 qui correspond à sa position :

$11010111 = 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

soit : $1 \times 128 + 1 \times 64 + 0 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 = 215$.

Les bits qui composent un octet sont numérotés de 0 à 7 à partir de la droite. Ceci est dû au fait que le bit le plus à droite a le poids 2^0. Mais attention, il y a là une source d'erreur fréquente car on numérote instinctivement à partir de 1. Cela implique que le bit 0 est le premier, le bit 1 est le second, etc.

Le plus grand nombre qu'on puisse obtenir sur 8 bits est 11111111, c'est-à-dire :

$128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$

Si l'on désire coder un nombre plus grand que 255, on utilisera deux octets ; il y aura un **octet de poids fort** et un **octet de poids faible**, le premier étant le prolongement à gauche du second.

Bit n°	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Valeur	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	octet de poids fort								*octet de poids faible*							

La valeur d'un nombre codé sur deux octets est égale à celle de l'octet de poids faible + 256 fois celle de l'octet de poids fort :

11110100	:	00101101
244 × 256	+	45
donc	=	62509

(Voir plus loin le paragraphe Grands nombres)

Dans le MSX, la memoire est organisée par groupes de huit bits, c'est pourquoi nous parlerons beaucoup d'octets, encore d'octets, toujours d'octets !

Evidemment, une suite de 0 et de 1 n'est ni très parlante, ni très facile à lire, et encore moins à mémoriser ! C'est pourquoi, pour nous simplifier la vie (!), nous allons introduire une nouvelle base.

Il est en effet pratique de diviser l'octet en deux groupes de 4 bits. Chaque groupe de 4 bits (**quartet**) peut compter de 0000 à 1111 binaire, soit de 0 à 15 en décimal. Une base nous permettant de compter de 0 à 15 sur un seul chiffre représenterait facilement un quartet : c'est la base 16 aussi appelée base **hexadécimale**.

En base 16, on dispose de 16 « chiffres » qui sont les 10 chiffres habituels (0 à 9), et les six premières lettres de l'alphabet A, B, C, D, E, F. A vaut 10, B vaut 11, ... F vaut 15. Voici une table de conversion binaire-hexadécimal-décimal :

Binaire	**Hexa**	**Décimal**
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15
10000	10	16
10001	11	17
10010	12	18
10011	13	19
10100	14	20

Un octet binaire pourra donc être écrit avec deux chiffres hexadécimaux.

Le système hexadécimal est précieux en programmation, car il se convertit facilement en binaire (chaque chiffre représente un quartet), mais est tout de même (un peu) plus «parlant» qu'une suite de 0 et de 1.

Bien entendu, un nombre sur deux octets prendra 4 chiffres hexadécimaux :

10110011 11011001 sera
B 3 D 9 = B3D9

Une convention d'écriture permet de s'y retrouver : un nombre hexadécimal sera toujours suivi de la lettre H.

Exemple : 0AH = 0A hexa = 10 décimal.

Ainsi, on ne confondra pas 10H (héxa) et 10 (décimal).

■ *Conversion entre bases*

□ La conversion binaire-hexadécimale ou hexadécimale-binaire se fait facilement en séparant l'octet en deux quartets, puis en se reportant à la table :

11001000 en binaire vaut
C 8 en hexadécimal

□ La conversion inverse est aussi facile :

A 8 en hexadécimal vaut
10101000 en binaire

□ La conversion binaire-décimal se fait à l'aide du principe de numeration de position.

$110b = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 4 + 2 = 6$

□ La conversion décimal-binaire se fait en retranchant successivement les puissances de 2 les plus élevées possibles ; celles qui ont pu être retranchées mettent à 1 le bit de poids correspondant. Cela paraît compliqué, mais en fait c'est très simple :

Soit à convertir 105 en binaire. 105 contient 64, mais pas 128 c'est-à-dire 2^6 mais pas 2^7, donc le bit 7 = 0, le bit 6 = 1.
105 − 64 = 41 ; 41 contient 32 (2^5) donc le bit 5 = 1 ;
41 − 32 = 9 ; 9 ne contient pas 16 donc le bit 4 est à 0 ;
9 contient 8 : le bit 3 est à 1 ; 9 − 8 = 1 donc (1 ne contient ni 4 ni 2) le bit 2 et le bit 1 sont à 0 ; il reste 1 : le bit 0 est à 1.

Nous obtenons donc :

N° bit	7	6	5	4	3	2	1	0
	0	1	1	0	1	0	0	1

☐ Les conversions décimal-hexadécimal se font en passant par le binaire ou par une table (cf. Annexe).

Pour finir ce paragraphe, voici deux autres définitions. On appelle **kilo-octet** (Ko) un groupe de 2^{10} octets. Un Ko vaut donc 1024 octets. On appelle **page** un groupe de 2^8 octets. Une page vaut donc 256 octets.

Il est indispensable d'avoir bien assimilé ces notions avant de poursuivre ; aussi nous vous proposons quelques exercices. FAITES-LES ! Un corrigé se trouve à la fin du livre en annexe 1.

Exercice 1.1 :

a) Combien vaut 10101010 en décimal ?
b) En hexadécimal ?
c) Combien vaut 7H en décimal ?
Même question avec FFH ?

Exercice 1.2 : Vérifier que le système hexadécimal est une numération positionnelle. On prendra l'exemple 3FCAH et on cherchera le poids de chaque chiffre.

Exercice 1.3 :

a) Combien y a-t-il de pages dans 1 Ko ?
b) Combien y a-t-il d'octets en hexadécimal dans une page ?
c) Et dans 1 Ko ?

ORGANISATION MÉMOIRE

■ *Cases mémoire*

La mémoire d'un ordinateur peut se comparer avec un pigeonnier dont les murs sont couverts de petites cases. Ces cases sont numérotées pour pouvoir les distinguer, et peuvent contenir un nombre. Chaque case peut contenir 8 bits (c'est-à-dire un octet). Il y aura donc dans chaque case mémoire (aussi appelée **emplacement mémoire**) un nombre compris entre 0 et 255 (0 et FFH). Dans le MSX, il y a au maximum 65536 cases mémoire différentiables en même temps. Leurs numéros vont par conséquent de 0 à F FFF H.

Exemple :	**N° case**	**contenu**	
	0400H	A3H	(= 10100011)
	0401H	30H	(= 00110000)

■ *Adresses*

Dans un village qui n'aurait qu'une rue, l'adresse et le numéro d'une maison se confondent. Dans le pigeonnier, c'est la même chose : l'adresse d'une case est son numéro.

Donc au lieu de dire « je mets A3H dans la case dont le N° est 400H » il est plus simple de dire : « je mets A3H à l'adresse 400H ».

Notre exemple précédent devient :

adresse	**contenu**
0400H	A3H
0401H	30H

Il est important de bien comprendre cette notion d'adresse, et de ne pas confondre le contenu (ici A3H) avec le contenant (ici l'adresse 0400H). Une adresse dont le N° prend 2 octets contient une donnée de 1 octet.

■ *Grands nombres*

Que faire si la donnée est constituée de 2 octets ? (par exemple une autre adresse) ?

Suivant les microprocesseurs, il y a deux façons de coder un mot de 16 bits :

Prenons le nombre C8E5H ; on appelle octet le plus significatif ou MSB *(Most Significant Byte)*, ou encore partie forte, celui qui a le poids le plus fort ; l'autre est l'octet le moins significatif ou LSB *(Least Significant Byte)*, ou encore partie faible.

C8	E5
octet	octet
le +	le −
signif.	signif.
MSB	LSB

La première façon de coder un nombre 16 bits est de ranger dans la case de numéro le plus bas l'octet le moins significatif, puis dans la suivante, l'octet le plus significatif (codage LSB-MSB) :

adresses	**contenu**
0000H	
LSB	E5
MSB	C8
FFFFH	

L'autre façon est l'inverse (codage MSB, LSB).

adresses	**contenu**
0000H	
MSB	C8
LSB	E5
FFFFH	

C'est le microprocesseur qui impose la façon de faire, par la manière dont il lit les données en mémoire. Avec le Z80 qui équipe votre MSX, les données 16 bits sont codées LSB, MSB.

■ *Nature des cases mémoire ; ROM-RAM*

Les cases mémoire sont de deux sortes. Les unes, numérotées de 8000H à FFFFH sur le MSX (pour un modèle 32 Ko), sont appelées **mémoires vives** ou RAM *(Random Access Memory)*. Elles sont à la disposition du programmeur ; on peut y déposer une information, la relire plus tard (sans la modifier), ou au contraire la modifier si on le désire. Elles perdent leur contenu si on coupe le courant ; c'est pourquoi votre programme s'efface, qu'il soit BASIC, ASSEMBLEUR ou autre, à l'extinction de l'appareil.

Les autres, numérotées de 0 à 7FFFH sont appelées **mémoires mortes** ou ROM *(Read Only Memory)*. Elles ont été fixées à la construction de l'appareil. On ne peut que lire leur contenu, jamais le modifier. Elles ne s'effacent pas en l'absence d'alimentation. Elles contiennent le programme de base (Moniteur, Editeur, BASIC) qui fait fonctionner le MSX.

On nomme **routine** un court fragment de programme ayant une fonction bien déterminée. Nous verrons plus loin comment utiliser des routines de la ROM dans vos programmes.

Il existe encore une autre sorte de case mémoire : il s'agit des registres internes du microprocesseur. En nombre limité (une vingtaine environ), ces registres n'ont pas d'adresse, mais sont représentés par des lettres. Ils constituent le cœur de l'ordinateur. C'est dans ces registres que s'effectuent toutes les opérations arithmétiques et logiques. Nous y reviendrons bien sûr en détail.

REPRÉSENTATION DE L'INFORMATION

Vous savez maintenant que la mémoire est composée de cases successives (adresses) les unes au-dessus des autres. Mais que contiennent-elles ? Toujours des octets !

Un octet en soi n'est rien d'autre qu'un nombre (de 0 à 255, 0 à FFH). Que peut bien représenter ce nombre ? Principalement trois choses :

■ *Un code opération*

Une **instruction** commande au microprocesseur d'exécuter une tâche élémentaire déterminée. Elle est composée d'un **code opération** (1 ou 2 octets) qui indique au microprocesseur la nature de l'opération à effectuer. Certains codes opération demandent 1 ou 2 octets supplémentaires qui contiennent l'**opérande** (donnée) sur lequel ils vont travailler. Les instructions occupent donc de 1 à 4 octets. Par exemple l'octet 01111000 (78H) commande au microprocesseur de charger le registre A avec le contenu du registre B. C'est une instruction sur un seul octet, il n'y a pas d'opérande. Chaque instruction possède une **mnémonique** c'est-à-dire un sigle permettant

d'identifier l'instruction et de s'en souvenir facilement. Le mnémonique de 01111000 est LD A,B (LD pour *LoaD*-charger).

Dans la pratique, vous n'aurez à vous souvenir que de la mnémonique, c'est l'ASSEMBLEUR qui se chargera de la traduire en octets compréhensibles par la machine lors de la phase d'assemblage (voir chapitre Assembleur).

- *Une donnée*

Deux cas principaux :

□ *L'octet représente un nombre*

Par exemple 00001010 (0AH) représente le nombre 10 décimal.
Ce nombre peut être considéré comme signé ou non signé suivant la nature du code opération qui opère sur lui. Ne vous affolez pas, nous y reviendrons en temps utile !

Exemple :	3E		0A
	code-op		opérande
	LD	A, 0A	

signifie charger l'accumulateur A avec la valeur 0AH. C'est une instruction sur deux octets.

□ *L'octet représente un signe alphanumérique*

On utilise pour les lettres, chiffres et signes un codage international appelé code ASCII. C'est un standard presque universellement adopte dans les micro-ordinateurs du marché. Toutefois il en existe d'autres, mais nous n'en parlerons pas ici. Vous trouverez en Annexe 2 une table des codes ASCII.

Par exemple : 01000001 (41H) représente la lettre A
00100001 (21H) représente le signe « ! »

- *Une adresse*

Nous avons vu qu'une adresse (comprise entre 0 et FFFFH) se code sur 16 bits. Il faudra donc 2 octets consécutifs pour la représenter.

Exemple :	3A		40 31
	code-op		adresse (opérande)
	LD	A, (3140)	

signifie charger l'accumulateur A avec le contenu de l'adresse 3140H. Attention, souvenez-vous : on trouve l'octet le moins significatif d'abord, donc 4031 et non 3140 en mémoire. Cette instruction est codée sur 3 octets : 1 pour le code opération, 2 pour l'opérande.

Comment savoir dans tout cela ce que représente un octet ? Cela dépend de sa place. Remarquez dans les trois exemples donnés plus haut qu'à chaque fois le code opération est différent. C'est lui qui détermine s'il y a ou non des octets opérandes dans l'instruction.

Le premier octet que reçoit le microprocesseur doit être un code opération. Suivant sa nature, celui-ci peut ou non demander 1 ou 2 octets opérandes. Ces opérandes peuvent être des données (on les appelle **données immédiates**), ou bien des adresses où il faudra aller chercher ces données. De cette façon, le microprocesseur peut aller d'instruction en instruction sans se « tromper d'octet ».

Supposons que l'instruction demande 2 octets opérande, comme l'exemple vu plus haut LD A, (3140) ; l'octet suivant sera considéré comme une instruction, et ainsi de suite. Si le microprocesseur tombe à ce moment sur un octet qui ne correspond pas à un code opération existant, il y aura « plantage » de la machine, c'est-à-dire un blocage dont on ne peut sortir qu'en éteignant l'ordinateur, perdant par ce fait programme et données. C'est ce qui arrive presque toujours si, à la suite d'une erreur de programmation, on tente de faire « exécuter » une donnée. Ces erreurs étant assez fréquentes chez le débutant, il vaut mieux sauver vos programmes sur cassettes avant la première exécution pour éviter d'avoir à les retaper en cas de plantage.

FONCTIONNEMENT SÉQUENTIEL

Récapitulons. Nous avons un bloc mémoire composé d'adresses successives, numérotées les unes après les autres. Dans ces adresses se trouvent rangés dans un ordre précis les octets qui constituent le programme. Le microprocesseur ira d'instruction en instruction, en allant des adresses les plus basses vers les plus hautes, jusqu'à ce qu'on lui indique la fin du programme. C'est ce qu'on appelle **fonctionnement séquentiel :** on fait une chose, puis, quand on a fini, on fait la suivante. Le Z80 ne peut penser à deux choses à la fois !

Nous verrons cependant que certaines instructions permettent une rupture de séquence, c'est-à-dire qu'elles provoquent un saut à une adresse qui n'est pas la suivante, mais qui se trouve ailleurs dans le programme. On peut ainsi former des boucles qui s'exécuteront plusieurs fois, et/ou sauter des parties non utiles à la suite de tests.

ALGORITHME, ORGANIGRAMME

Ceci montre la nécessité de structurer le programme.

Pour effectuer une tâche en langage machine, il faut d'abord bien poser le problème, puis concevoir le principe de résolution : c'est l'**algorithme.**

Exemple : chercher si une suite d'octets (on dit une TABLE de données) contient ou non la valeur 42H. (C'est le code ASCII de la lettre B). L'algorithme sera :

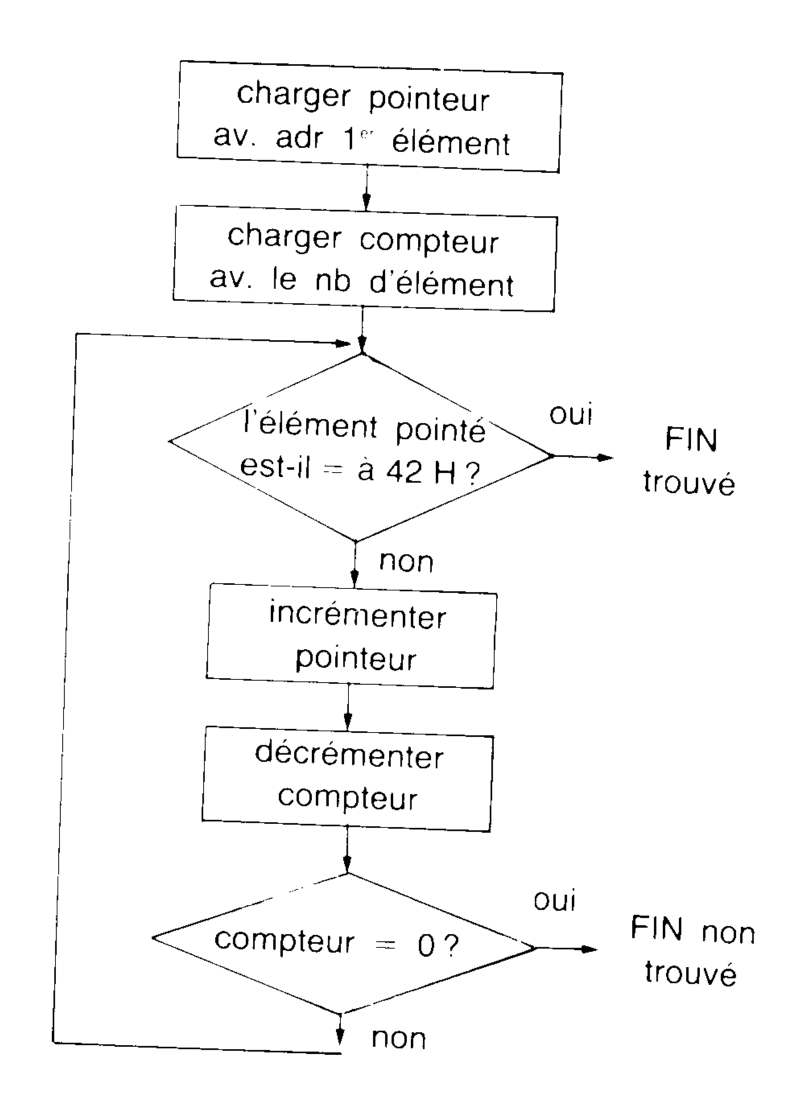
— créer un « pointeur » qui « montre du doigt » le début de la table,
— créer un compteur qui comptera les éléments comparés,
— comparer l'élément montré à la valeur 42H,

— si l'égalité est vérifiée, on saute à la fin du programme,
— si l'égalité n'est pas vérifiée, incrémenter (augmenter de 1) le pointeur pour qu'il « montre » l'élément suivant,
— comparer de nouveau, et ainsi de suite jusqu'à ce qu'on ait trouvé l'égalité, ou qu'on ait parcouru toute la table.

Il est important de bien saisir le principe du pointeur : cela pourra être un registre interne du Z80 ou bien 2 cases mémoire consécutives qui contiendront l'adresse effective de l'élément cherché. En incrémentant ou décrémentant (diminuer de 1) le pointeur, on obtient respectivement l'adresse suivante ou précédente, donc l'élément suivant ou précédent. Nous reverrons ceci dans le chapitre sur les modes d'adressage (adressage indirect).

Une fois l'algorithme au point, on peut faire (c'est recommandé) un **organigramme.**

C'est un dessin du « chemin » utilisé pour obtenir le résultat, avec toutes les étapes intermédiaires. Voici l'organigramme correspondant à l'algorithme ci-dessus :



Il ne reste plus qu'à transposer chaque bloc avec les instructions adéquates. Vous ne savez pas encore le faire, mais patience !

2

Introduction à l'Assembleur

Comme nous l'avons signalé dans l'introduction, il existe une confusion au sujet du mot ASSEMBLEUR. Le microprocesseur ne comprend, nous l'avons vu, que des suites d'octets binaires du type 00011010,11101110, 00001000, etc. C'est ce qu'on appelle le **code machine**. La difficulté de manipuler des valeurs binaires qui se ressemblent toutes et sont difficiles à lire, engendrerait un risque d'erreurs énorme si l'on devait programmer directement en code machine.

On utilise donc généralement une représentation (un peu) plus parlante : c'est le code **mnémonique**. Les mnémoniques sont des abréviations de mots anglais décrivant l'action des instructions. Bien entendu, ces mnémoniques ne sont pas comprises par le microprocesseur et il est nécessaire de les traduire en code machine pour les exécuter.

Il est possible de faire la traduction soi-même en entrant les octets un à un dans des DATA en BASIC (sous forme hexadécimale pour une manipulation plus facile). Malgré tout, cette méthode reste limitée à des routines courtes, car les erreurs sont difficilement visibles, et le temps d'écriture assez long : il faut rechercher un à un les codes des instructions voulues, puis les combiner dans le bon ordre, calculer à la main les adresses de branchement.

Il était logique de confier cette fastidieuse traduction à l'ordinateur, qui ne renâcle jamais à la tâche ! On a donc écrit un programme appelé ASSEMBLEUR, qui effectue la traduction mnémonique-code machine. L'ASSEMBLEUR est, par conséquent, un programme traducteur et, par extension, il désigne aussi la programmation en langage machine.

Le programme écrit en mnémoniques constitue un texte (peu littéraire !) ; on l'appelle **programme source** ou simplement Source. Il obéit à certaines règles de syntaxe que nous détaillerons plus loin.

L'**assemblage**, c'est-à-dire la traduction du programme source générera un **programme objet** en code machine directement exécutable. L'écriture du programme source est grandement facilitée par un **éditeur** de texte qui permet de rentrer les instructions aisément, de retrouver tel ou tel passage, de corriger les erreurs, et d'insérer des commentaires. Il est rare que le programme fonctionne correctement du premier coup. Les erreurs de syntaxe sont détectées par l' ASSEMBLEUR au moment de l'assemblage, mais non les erreurs de **logique**, ni l'oubli d'une ou plusieurs instructions. On utilise alors un **moniteur** pour le *debugging* c'est-à-dire la « chasse aux fautes ». Le moniteur admet un nombre varié de commandes, comme l'exécution pas à pas, la visualisation des registres, la recherche d'une suite d'octets en mémoire, leur modification, etc. Il est grandement recommandé (surtout aux débutants) de sauver sur cassette ou sur disquette le programme source AVANT de lancer la première exécution à l'aide du moniteur, ce qui évitera de tout perdre (Source et Objet) en cas de « plantage ».

Un tel Moniteur Editeur Assembleur existe sous le nom de ODIN, édité par la Société LORICIELS. Il s'agit d'un ASSEMBLEUR symbolique à double passe, d'un très haut niveau. Bien que les conventions puissent varier légèrement d'un ASSEMBLEUR à l'autre, nous prendrons les notations et la syntaxe de ODIN tout au long de cet ouvrage. Nous vous conseillons de vous en procurer la cassette chez votre revendeur. Cependant, cela n'est pas indispensable et vous pourrez travailler avec un autre ASSEMBLEUR.

CONVENTIONS DE NOTATIONS

- Nous serons souvent amenés dans le courant de ce livre, ainsi que vous-même dans vos programmes, à parler du contenu d'une adresse ou d'un registre.

  — Le contenu d'un registre se notera par le nom de ce registre :

  HL désigne en programmation le contenu du double registre HL.

  Il n'y a pas d'ambiguïté possible, car le registre lui-même est un ensemble de connexions physiques fixées par construction, et qu'aucun programme ne pourra jamais modifier.

  — Le contenu des cases mémoire sera noté par les parenthèses ( ) qui signifieront « le contenu de ».

  Exemple : 53 désigne la valeur 53
  (53) désigne le contenu de l'adresse 53

  BC désigne le contenu du double registre BC.

  — (BC) désigne donc le CONTENU DU CONTENU du double registre BC.

  Attention à ce dernier point ! Si BC contient 3000, (BC) désigne le contenu de l'adresse 3000. C'est ce qu'on appelle une **indirection** (voir Mode d'adressage indirect au chapitre 4).

- L'ASSEMBLEUR accepte les valeurs numériques en décimal et en hexadécimal ; pour les distinguer, une valeur hexadécimale est toujours suivie de la lettre H. La lettre D est

facultative et désigne les valeurs décimales. ODIN accepte aussi le système octal dont nous n'avons pas parlé, car il n'est plus guère utilisé ; le suffixe est alors O.

Les apostrophes " désignent le code ASCII de la lettre qu'elles encadrent : "A" = 65 (=41H).

- Des conventions supplémentaires, particulières à ce livre seront données au chapitre 5 Jeu d'instructions. Elles ne serviront qu'à alléger la présentation de ce chapitre, et ne doivent pas être employées dans les programmes.

SYNTAXE GÉNÉRALE DE L'ASSEMBLEUR

- *Symboles*

Un ASSEMBLEUR symbolique admet les valeurs numériques représentées par des SYMBOLES (ou ETIQUETTES, ou encore LABELS). Une étiquette est un mot d'au plus 6 caractères, dont le premier doit être une lettre ou un signe de code ASCII supérieur à 63, mais jamais un chiffre.

Une étiquette peut être définie de deux façons :

— Mise dans le champ étiquette d'une instruction (voir ci-dessous), elle prend la valeur de l'adresse de cette instruction.

— Elle peut être définie par la directive EQU (voir plus loin).

La possibilité d'utiliser des labels apporte une très grande simplification à l'écriture des programmes.

Pour que l'ASSEMBLEUR puisse faire la différence entre une étiquette et un nombre hexadécimal sans ambiguïté possible, les nombres hexadécimaux dont le premier "chiffre" est A,B,C,D,E, ou F doivent être précédés d'un 0 :

0BACH est le nombre hexa BAC ;

BACH est l'étiquette BACH (pourquoi pas ?).

- *Format d'une ligne Assembleur*

Les lignes ASSEMBLEUR (du programme source) obéissent à des règles syntaxiques précises. Elles doivent comprendre un certain nombre de **champs**, c'est-à-dire de parties différentes. Il y a 5 champs dont 1 ou 2, suivant la nature de l'instruction, sont obligatoires ; les autres sont optionnels. Ces champs doivent être séparés par au moins un espace blanc, ou mieux, par un caractère de tabulation (TAB).

Examinons les différents champs :

□ *Le numéro de ligne*

Chaque ligne commence par un numéro. Mais à la différence du BASIC, ce numéro ne figurera pas dans le programme objet. Il ne sert qu'à faciliter la manipulation du texte source : recherche de lignes, insertions, suppressions, corrections, etc.

□ *Le champ étiquette* (ou label, ou encore symbole)

Si vous désirez donner un nom à l'adresse d'une instruction, écrivez ce nom (suivant les règles énoncées au paragraphe Symboles) dans le champ étiquette de la ligne concernée. Vous pourrez utiliser cette étiquette en remplacement de l'adresse à chaque fois que vous en aurez besoin. Mieux encore, ODIN accepte dans le champ opérande une étiquette définie PLUS LOIN dans le programme source. Cela est très pratique pour l'écriture des branchements en aval. A l'assemblage, au cours de la 1re passe, ODIN constitue une table de toutes les étiquettes définies dans le Source ; ce qui lui permet, au cours de la 2e passe, de remplacer dans le code objet ces étiquettes par leur valeur.

□ *Le champ code opération*

C'est le seul qui soit toujours obligatoire (sauf ligne de commentaire pur). Il doit contenir le **littéral** de la mnémonique de l'instruction, à l'exclusion de la ou des opérandes. Si, par exemple, l'instruction est LD A,C, le littéral est LD ; de même, pour JP NC,9000H, le littéral est JP.

On peut aussi mettre dans le champ code opération une directive d'assemblage, qui ne générera pas de code objet, mais qui agira sur le déroulement de l'assemblage ou générera des octets quelconques (voir Directives).

□ *Le champ opérandes*

Les opérandes indiquent quels sont les registres, adresses ou données concernés par l'instruction. La façon de les écrire indique le mode d'adressage désiré (voir Conventions d'écriture, modes d'adressage).

Les valeurs numériques (adresses, constantes, etc.) peuvent être représentées par des symboles, comme expliqué plus haut. Elles peuvent aussi être des expressions arithmétiques, que l'ASSEMBLEUR effectuera au moment de l'assemblage. Enfin, un signe alphanumérique entre apostrophes représente le code ASCII de celui-ci.

□ *Le champ commentaire*

Ce champ vous permet de faire figurer des commentaires dans le texte du Source. Même si un programme vous paraît très clair en lui-même au moment de son écriture, il est presque sûr que quelqu'un d'autre n'y comprendra rien ; vous risquez vous-même, si vous y revenez quelque temps plus tard pour le modifier, de ne plus vous y retrouver. Il est donc nécessaire de commenter votre programme judicieusement.

Le champ commentaire doit commencer par un point-virgule (;). Si le commentaire est long, vous pouvez entrer une ligne spéciale pour lui. Le point-virgule suivra alors le numéro de ligne (après au moins un espace).

Le format général d'une ligne est donc :

[étiquette][TAB code-op [opérandes]][TAB][;commentaires]

où [ ] désignent les termes optionnels.

Exemple :

```
N° étiq. code-op opérandes comment.
10 debut    LD     A,'X'      ;met le code ASCII de X
20 ;dans le registre A
```

■ *Directives*

Les directives d'assemblage sont des pseudo-instructions utilisées pour agir sur l'assemblage lui-même. Elles ne génèrent pas toujours de code objet.

☐ ORG indique à l'assembleur à quelle adresse implanter le code objet.

Exemple :

```
ROUTN1 ORG    6000H
       ---
       routine 1
       ---
ROUTN2 ORG    ROUTN1+200H
```

☐ END indique la fin du texte à assembler. On met souvent en opérande l'adresse de lancement du programme (qui peut très bien ne pas être la plus basse utilisée par le programme ; c'est un bon moyen de ne pas l'oublier !).

☐ EQU assigne une valeur à une étiquette. Cette valeur ne pourra plus être changée dans la suite du programme source sous peine d'une erreur du type « définition multiple ». Les étiquettes définies par EQU, contrairement aux autres doivent être définies avant leur utilisation.

Exemple :

```
ICI    EQU    7890H
LA     EQU    ICI-50H
STOP   EQU    ' '      ;code ASCII de espace
```

De nombreuses autres directives peuvent être employées (ODIN en admet une dizaine). Voyez la documentation de votre ASSEMBLEUR pour les détails.

3

Arithmétique et logique

Nous avons expliqué au premier chapitre comment les informations sont codées en octets. Il va falloir maintenant apprendre à manipuler ces octets.

Trois sortes d'opérations sont possibles sur des valeurs binaires : les opérations arithmétiques, les opérations logiques et les opérations de décalage et rotation. On peut considérer les décalages et les rotations comme faisant partie des opérations logiques ; c'est ce que nous ferons ici.

OPÉRATIONS ARITHMÉTIQUES

L'arithmétique binaire concernant les entiers positifs est très simple et fonctionne de manière similaire à l'arithmétique décimale, en gardant à l'esprit que l'on ne dispose que de 2 chiffres, 0 et 1 :

$$0 + 0 = 0$$
$$1 + 0 = 1$$
$$1 + 1 = (1)0$$

ou (1) représente la retenue (« je pose 0 et je retiens 1 »)

et bien sûr :

$$1 + 1 + 1 = (1)1$$

Une addition sur 8 bits s'effectue normalement de droite à gauche, en tenant compte à chaque fois de la retenue, s'il y en a une.

Exemple :

	10001100	(=140)
+	00010001	(=17)
=	10011101	(=157)

	00001011	(=11)
+	00100101	(=37)
=	00110000	(=48)

Exercice 3.1 : Effectuez les opérations suivantes :

	01010101			10001100
+	00110100		+	00001101
=			=	

Un problème est evident : que se passe-t-il s'il y a une retenue au-delà du bit le plus à gauche (bit 7) ? Le résultat ne tient plus sur 8 bits, ce qui est gênant, car c'est la capacité maximale de la plupart des cases mémoire et des registres. Le résultat risque donc d'être faux. Heureusement, on dispose dans tous les microprocesseurs, et dans le Z80 en particulier, d'un indicateur special appelé CARRY (Retenue). Entre autres rôles, il prend la valeur du 9ᵉ bit lors d'une addition, ce qui permet de ne pas perdre d'information. Carry prend donc la valeur 1 si il y a eu dépassement de capacité, 0 sinon.

Exemple :

		10110010
+		11000011
=	(1)	01110101

Carry prend la valeur 1, puisqu'il y a report au-delà du bit 7.

Vous savez maintenant additionner deux entiers positifs.

Pour faire une soustraction, le microprocesseur additionne en fait l'opposé (A − B = A + (− B)). Vous aurez donc besoin de manipuler les entiers négatifs ; voyons comment les représenter. Plusieurs façons sont envisageables. Le signe d'un nombre étant soit + soit −, il est logique d'affecter à un bit de l'octet le rôle de signe. On peut par exemple choisir le bit 7 ; s'il est à 1, il indiquera un entier négatif, s'il est à 0, il indiquera un entier positif.

L'avantage de cette méthode est que le nombre ne dépasse pas 8 bits, mais l'inconvénient est qu'il ne reste plus que 7 bits pour la valeur absolue. On ne pourra donc coder sur un octet que les entiers compris entre − 127 et + 127.

Exemple :

00001110	= 14
10001110	= −14

Un autre inconvénient de cette représentation est que la somme de deux entiers opposés n'est pas nulle :

	00010001	(17)
+	10010001	(−17)
=	10100010	(−34)

Pour cette raison, on a cherché une autre méthode : c'est la complémentation. La méthode définitive sera le complément à 2 ; mais pour l'introduire, il est plus simple d'expliquer ce qu'est le complément à 1.

En complément à 1, un entier négatif s'écrit en inversant tous les bits de sa valeur absolue :

9 s'écrit 00001001
−9 s'écrit 11110110

Le bit 7 représente toujours le signe. S'il est à 1, on a affaire à un entier négatif. Pour trouver sa valeur absolue, il faut inverser (complémenter) tous ses bits.

L'ennui, c'est que l'arithmétique ne marche toujours pas !

Exercice 3.2 : Effectuer les opérations suivantes en représentation en complément à 1 ; vérifiez que le résultat est incorrect : −4+5 ; 2−6 (c'est-à-dire 2+(−6)).

Nous emploierons donc la représentation en complément à 2 : le complément à 2 d'un entier est simplement son complément à 1 augmenté de 1.

```
 9 s'écrit  00001001
−9 s'écrit  11110110 (complément à 1 de 9)
          +        1
          ----------
          = 11110111 (complément à 2 de 9)
```

De même que précédemment,le bit 7 indique le signe du nombre, et sera à 1 pour un entier négatif ou à 0 pour un entier positif. Par cette méthode, on peut coder sur un octet les entiers de −128 à +127. On trouve la valeur absolue d'un entier négatif en effectuant une nouvelle complémentation à 2 :

Soit le nombre 11110111 ; Il est négatif puisqu'il commence à gauche par un 1. Calculons sa valeur absolue :

```
son complément à 1 est : 00001000
son complément à 2 est : 00001000
                       +        1
                         --------
                         00001001  = 9
```

Le nombre original 11110111 était donc −9. La complémentation à 2 agit exactement comme un changement de signe. Si l'on fait un nombre pair de fois la complémentation à 2, on retrouve l'original, alors qu'un nombre impair de fois donne l'opposé de l'original.

Vous savez maintenant soustraire deux nombres binaires : on ajoute au premier le complément à deux du second.

Exercice 3.3 : Effectuez les opérations suivantes et vérifiez que le résultat est correct (ne pas tenir compte de Carry pour le résultat) : 5−2 ; 4+6 ; 3−7 ; −3−3.

Cette représentation fonctionne donc bien. A moins que... Oui, il y a un mais ! Effectuons 120 + 10 :

```
  01111000     (=120)
+ 00001010     (=10)
  --------
= 10000010     (=−125)
```

Nous avons là une erreur due au changement de signe, et à un dépassement de capacité interne : les entiers positifs vont de 0 à 127, les négatifs de 0 à −128. Si l'on tente de faire 120+10, le résultat (130) ne tient pas sur 8 bits signés : il y a retenue interne du bit 6 sur le bit de signe, d'où changement « accidentel » du signe et erreur du résultat. C'est ce qu'on appelle un **débordement**. Ce débordement se produira également en additionnant de grands entiers négatifs.

Cependant, comme si tout cela n'était pas assez compliqué, le report du bit 6 vers le bit 7 n'implique pas forcément un débordement (donc erreur).

Examinons les cas suivants :

```
          11111111    (=−1)
        + 11111111    (=−1)
=     (1) 11111110    (=−2)
```

Il y a bien eu report interne du bit 6 vers le bit 7, mais aussi du bit 7 vers Carry. Dans ce cas, le résultat est correct, et il n'y a pas de débordement.

Faisons maintenant :

```
          11000000    (=−64)
        + 10111111    (=−65)
=     (1) 01111111    (=+127)
```

Il y a eu report vers Carry *SANS* report interne du bit 6 vers le bit 7. Il y a débordement.

De même que la retenue du bit 7 vers l'extérieur affecte l'indicateur Carry, le débordement affecte un indicateur appelé V.

Pour résumer, il y a débordement, donc erreur et mise à 1 de V s'il y a report du bit 6 vers le bit 7 sans report vers l'extérieur, ou s'il y a report vers l'extérieur sans report interne.

S'il n'y a pas de débordement, on ne prend pas en compte la valeur de Carry pour le résultat.

Vous n'avez (heureusement) pas besoin de retenir tout ceci pour l'instant ; souvenez-vous simplement que V=1 indique une erreur en arithmétique signée, et qu'on ne s'occupe pas de Carry.

Quand manipule-t-on des nombres négatifs ? Principalement dans les instructions de saut relatif (voir chapitre Modes d'adressages).

Jusqu'ici, nous sommes limités aux entiers positifs et négatifs compris entre 127 et −128. Si les valeurs sont plus grandes, on peut opérer sur deux ou trois octets, en gardant le même principe. Une autre représentation appelée **virgule flottante** permet de représenter les nombres décimaux, positifs ou négatifs, en utilisant plusieurs octets (signé, mantisse, exposant). Nous ne l'utiliserons pas ici ; les calculs y sont plus complexes, mais les opérations de base sont les mêmes.

Par contre, il faut signaler une dernière représentation qu'on utilise quand on a besoin d'une grande précision. En effet, en représentation binaire signée, quel que soit le nombre d'octets adopté, il y a toujours une capacité maximale. Si elle est dépassée au cours d'un

calcul, il y a perte d'information dans le résultat final ; généralement les chiffres les moins significatifs sont tronqués, ce qui n'est pas toujours admissible, par exemple en comptabilité où chaque centime compte. On utilise alors la représentation DCB.

REPRÉSENTATION DCB

DCB signifie Binaire Code Décimal. Le principe est de coder chaque CHIFFRE décimal séparément, et d'utiliser autant de bits que nécessaire pour coder tout le nombre. Pour pouvoir représenter les chiffres de 0 à 9, il faut 4 bits (soit un quartet). Trois bits ne pourraient compter que jusqu'à 7. Quatre bits donnent 16 combinaisons, ce qui veut dire que 6 seront inutilisées en DCB. Puisqu'il nous faut 4 bits par chiffre décimal, on peut en ranger 2 dans un octet. Voici une table DCB :

Quartet	**Chiffre décimal**	**Quartet**	**Chiffre décimal**
0000	0	1000	8
0001	1	1001	9
0010	2	1010	inutil.
0011	3	1011	inutil.
0100	4	1100	inutil.
0101	5	1101	inutil.
0110	6	1110	inutil.
0111	7	1111	inutil.

Exemple : 00000111
vaut 0 7
00111001
vaut 3 9

Essayons une addition :

```
  00000011   (=03)
 +10000110   (=86)
 =10001001   (=89)
```

Il semble que cela fonctionne. Cependant, un problème se pose : qu'arrivera-t-il si au cours d'un calcul, l'un des 6 codes inutilisés est obtenu ? Une erreur se produira, car aucun chiffre décimal ne correspondra à ce code.

Exemple :

```
  00000111   (=07)
 +00001000   (=08)
 =00001111   (=0?)
```

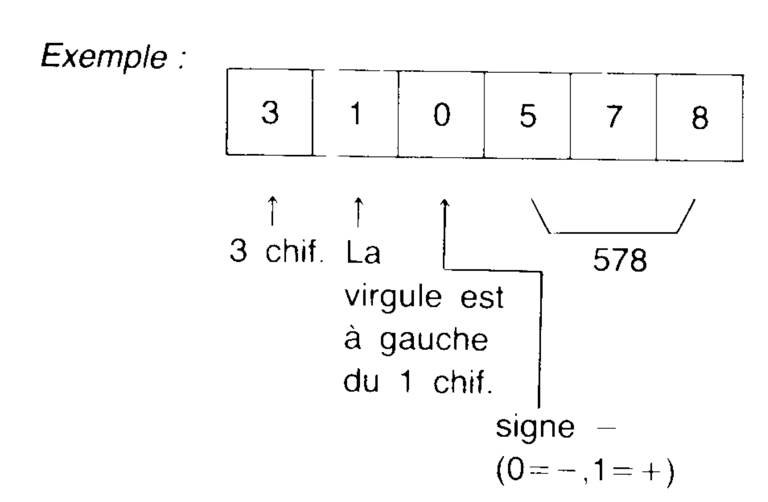
Le quartet de droite ne correspond à aucun code DCB. Une correction est donc nécessaire. C'est pourquoi il existe une instruction spéciale (DAA = ajustement décimal) qui, après une opération, corrige le résultat pour n'obtenir que des codes admis en DCB. Cette instruction agit de façon assez compliquée, bien que le principe en soit simple : il faut « sauter » les 6

codes interdits, c'est-à-dire ajouter 6 au(x) quartet(s) comportant un code interdit, tout en gérant la retenue entre quartet faible et quartet fort. Ceci pour une addition, car pour une soustraction, les choses sont différentes... De toute façon, DAA se débrouille toute seule, corrige ce qu'il faut, et le résultat est correct !

Souvenez-vous simplement qu'en DCB, un octet compte de 0 à 99, que les octets ne sont pas signés, et que Carry=1 si le résultat est supérieur à 99.

Nous avons dit que pour coder de grands nombres en DCB, on utilise autant d'octets que nécessaire : plusieurs conventions sont possibles, mais on fait en général précéder le nombre DCB lui-même d'un ou deux octets qui indiquent le nombre de chiffres, le signe éventuel, et/ou la place de la virgule (si elle existe). On voit qu'on peut ainsi coder tous les nombres décimaux.

Exemple :



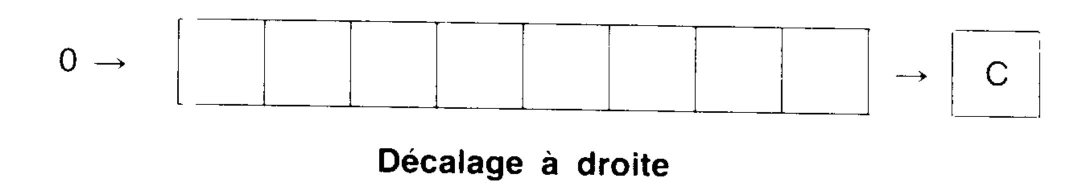
Nous avons donc une représentation de −57,8. Nous aurons l'occasion de donner des exemples plus précis d'utilisation du DCB.

Exercice 3.4 : Traduisez en DCB les nombres suivants : 54,97,12,05,114. Ce dernier tient-il sur un octet ?

OPÉRATIONS LOGIQUES

■ *Décalages et rotations*

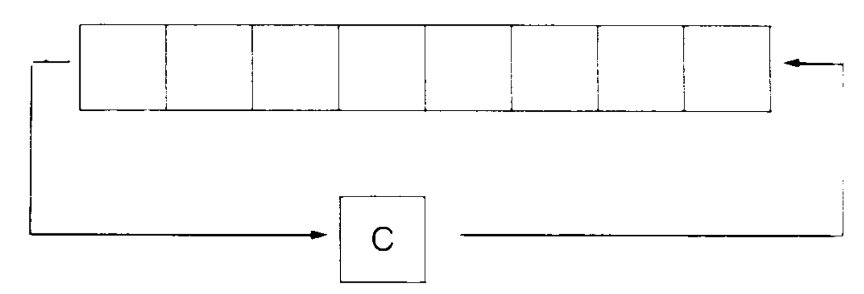
Le décalage d'un octet est une translation de tous ses bits vers la droite ou vers la gauche, l'un prenant la place de l'autre ; il entre un 0 d'un côté, et le bit qui « sort » tombe dans « indicateur Carry ».



Décalage à droite

Il existe plusieurs sortes de décalages, à gauche ou à droite suivant la nature du bit entrant (voir Jeu d'instructions).

Une rotation utilise le même principe, mais le bit entrant vient de Carry :



Rotation à gauche avec Carry

Il existe également plusieurs sortes de rotations avec ou sans Carry (le bit entrant peut aussi être celui qui sort de l'autre côté). Voyez le Jeu d'instructions pour plus de détails.

■ *AND, OR, XOR*

Les trois opérations AND, OR, XOR sont purement des opérations logiques ; ce sont les seules présentes sur le Z80, bien qu'il en existe d'autres (NAND, équivalence, etc.). Elles sont utiles pour les manipulations de bits à l'intérieur des octets.

□ *AND*

AND effectue le ET logique, bit à bit, entre deux octets. Le ET logique se définit comme suit : soit deux bits A et B ; A ET B vaut 1 si A et B valent 1 ensemble, 0 dans les autres cas :

A	B	A ET B
0	0	0
1	0	0
0	1	0
1	1	1

Table de vérité de AND

Donc par exemple :

```
      10011110
AND   00101101
 =    00001100
```

Exercice 3.5 : Calculez 10111001 AND 00111101 :

Remarquez que [octet] AND 11111111 ne change pas l'octet, et que [octet] AND 00000000 donne 00000000 quel que soit l'octet.

AND peut être utilisé pour forcer à 0 un bit quelconque d'un octet : [octet] AND 11110111 force à 0 le bit 3 de [octet] et laisse les autres intacts.

□ *OR*

OR effectue le OU logique, bit à bit entre deux octets. Soit deux bits A et B. Le résultat de A OU B vaut 1 si au moins l'une des deux opérandes vaut 1 :

A	B	A OU B
0	0	0
1	0	1
0	1	1
1	1	1

Table de vérité de OR

Donc par exemple :

```
      11100110
OR    00101001
=     11101111
```

Exercice 3.6 : Calculez 00010111 OR 10101011.

Remarquez que [octet] OR 00000000 ne change pas [octet], et surtout que [octet] OR 11111111 donne 11111111 quel que soit [octet].

OR peut être utilisé pour forcer à 1 un bit quelconque d'un octet : [octet] OR 00000010 force à 1 le bit 1 de [octet].

□ *XOR*

XOR effectue le ou exclusif, bit à bit entre deux octets. Soit deux bits A et B. A XOR B vaut 1 si un seul des deux bits A et B vaut 1 mais pas les deux :

A	B	A XOR B
0	0	0
1	0	1
0	1	1
1	1	0

Table de vérité de XOR

Donc par exemple :

```
       11001010
XOR    01011001
=      10010011
```

Exercice 3.7 : Soit l'octet 10111001 ; quelle(s) opération(s) doit-on effectuer pour obtenir 00111110 ?

4

Organisation interne du Z80

ARCHITECTURE GÉNÉRALE

Un microprocesseur est un circuit extrêmement compliqué. Pour apprendre à le programmer, il ne sera pas nécessaire (heureusement !) de comprendre en détail son fonctionnement électrique. Seule une représentation symbolique et simplifiée suffira à décrire ce fonctionnement. Si vous désirez en savoir plus à ce niveau, reportez-vous à une documentation « hard » sur le Z80. C'est en effet un Z80 qui équipe votre MSX. Il s'agit d'un microprocesseur 8 bits, c'est-à-dire qu'il traite des mots de 8 bits de longueur. Il en résulte que la mémoire elle aussi est organisée en octets, comme nous l'avons vu au chapitre précédent.

Le microprocesseur (MPU, *MicroProcessor Unit*) est composé d'une unité arithmétique et logique (UAL) qui effectue les calculs, de registres internes qui contiennent les données à traiter, d'un décodeur d'instruction, et d'une unité de commande qui séquence le système.

Pour communiquer avec l'extérieur, le MPU dispose de trois ensembles de lignes électriques appelés BUS.

— Le BUS de DONNEES est composé de 8 lignes ; il pourra donc véhiculer 8 bits en même temps. Il transportera les données du MPU vers la mémoire (RAM) ou vers un boîtier d'entrée/sortie qui communique avec un périphérique (écran, imprimante, cassettes, etc.). Le BUS de DONNEES peut aussi acheminer les données dans l'autre sens, vers le MPU, par exemple quand on lit une donnée en mémoire (ROM ou RAM). Il est donc bidirectionnel.

— Le BUS d'ADRESSES est composé de seize lignes. Il achemine vers les boîtiers mémoire externes, l'adresse, générée par le MPU, qui spécifie où lire et où ranger les données transportées par le BUS de DONNEES. Puisqu'il transporte des adresses, ce BUS est forcément sur 16 bits. Il est unidirectionnel.

— Le BUS de COMMANDE transporte les différents signaux de synchronisation de l'ensemble.

Une horloge précise est enfin nécessaire pour faire « tourner » le MPU ; un quartz à 3,7 MHZ remplit ce rôle.

REGISTRES INTERNES

A l'intérieur du MPU, le programmeur n'a accès qu'aux REGISTRES INTERNES. Ce sont des cases mémoire particulières qui contiennent un ou deux octets. On leur donne des noms plutôt que des adresses ; il y a au total 21 registres accessibles dans le Z80 :

A, B, C, D, E, F, H, L, IX, IY, SP, I, R, et les doubles A' à L'.

Certains de ces registres peuvent être réunis deux à deux pour former des registres 16 bits : AF, BC, DE, HL.

Registres 8 bits

A	F
B	C
D	E
H	L

Registres 16 bits

IX
IY
SP
PC

A'	B'
B'	C'
D'	E'
H'	L'

Registres spéciaux

I
R

Remarquez que les registres 8 bits ont un nom d'une seule lettre, et que ceux de 16 bits ont un nom de 2 lettres.

Examinons tous ces registres et leurs utilisations.

— Le registre A ou **accumulateur** (8 bits) a un rôle prépondérant : c'est sur lui que porteront les opérations arithmétiques et logiques. Nombre d'instructions ne concernent que l'accumulateur. L'association AF est à utiliser avec précaution. Ce n'est pas un véritable double registre (voir plus loin le registre F).

Exemple : LD A,42H charge la valeur 42H dans A.
CP B compare B à A. A n'est pas spécifié, car une comparaison opère toujours sur lui.

— Les registres B, C, D, E, H, L (8 bits) sont utilisés pour stocker temporairement des données destinées à être traitées par l'accumulateur. Le temps d'accès à ces registres est bien inférieur à celui de la mémoire externe, d'où l'intérêt de bien les utiliser. Ils sont équivalents entre eux avec une exception pour B sur qui agit une instruction particulière comme compteur de boucle (DJNZ). En association, HL a un rôle important : il est le double accumulateur (même rôle que A mais sur 16 bits), ou bien est souvent utilisé comme pointeur d'adresse (adressage indirect).

Exemple : LD C,20H charge la valeur 20H dans C.
LD HL,4000H charge la valeur 4000H dans HL (40H dans H,00 dans L)
LD B,(HL) charge B avec le contenu de l'adresse pointée par HL (adressage indirect). Ici B prendra la valeur qui se trouve à l'adresse 4000H.

— Les registres A', B', C', D', E', F', H', L, sont exactement symétriques. Ils prennent la place de A, B, C, D, E, F, H, L grâce aux instructions EX et EXX. Ils constituent la 2ᵉ banque de registres. On travaille avec l'une ou l'autre des 2 banques de registres, jamais avec les deux en même temps. La banque inutilisee peut conserver temporairement des données. Leur rappel sera plus rapide que si on les stocke en mémoire.

— Les registres IX et IY (16 bits) servent d'index (I pour INDEX), pour accéder à une donnée. En général, ils contiennent une adresse, à laquelle on ajoute, ou retranche, un déplacement pour trouver la donnée cherchée.

Exemple : LD IX,5000 charge 5000H dans IX
LD E, (IX+5) charge E avec le contenu de l'adresse 5005H.

— Le registre I n'est pas utilisé dans le système MSX.

— Le registre R contient une adresse qui change sans cesse et qui assure le « rafraîchissement » des mémoires vives externes. On ne doit pas l'utiliser en programmation, sauf cas spécial comme par exemple générer un nombre pseudo-aléatoire.

— Le registre SP est le pointeur de pile (*stack pointer*). Nous expliquerons son rôle plus loin. Il est à utiliser avec de grandes précautions.

— Le registre F (*flag*-drapeau) n'est pas un registre de travail. Chacun des bits qui le composent est un indicateur qui, suivant son état (0 ou 1), renseigne sur le résultat de certaines opérations. Nous étudierons le rôle exact de ces indicateurs dans le prochain paragraphe.

Il existe encore un registre que nous n'avons pas cité, car il n'est pas directement accessible au programmeur. Il s'agit du registre PC (*Programm Counter* — compteur programme). Il contient à chaque instant l'adresse de la prochaine instruction à exécuter. C'est un registre important : en s'incrémentant automatiquement à chaque instruction du nombre d'octets qui compose celle-ci, il construit le séquencement du programme, tout en évitant au MPU d'exécuter une donnée. Il est toutefois modifié par les instructions de saut (attention où vous sautez !). Celles-ci mettent simplement dans PC l'adresse opérande du saut.

Exemple : JP 6000 provoquera la poursuite du programme à l'adresse 6000H.

LES INDICATEURS (registre F)

Le registre F est sur 8 bits dont 6 sont significatifs et constituent les indicateurs C,N,P/V,H,Z,S.

	Registre F
Indicateur	S Z ? H ? P/V N C
Bit No	7 6 5 4 3 2 1 0

Nous allons étudier les différents indicateurs, et leurs rôles. C'est un point très important dans l'apprentissage du langage machine. Les indicateurs doivent être bien compris, car sur eux reposent les instructions conditionnelles (tests).

- *L'indicateur C* (carry — retenue arithmétique)

  La carry a un double rôle : d'une part, elle indique si une opération arithmétique a engendré une retenue vers le 9ᵉ bit; d'autre part, elle sert de 9ᵉ bit dans les opérations de décalage et de rotation (n'oubliez pas que le 9ᵉ bit est le bit numéro 8 puisque le premier bit est numéroté bit 0). Il faut garder à l'esprit ces deux aspects qui peuvent suivant les cas, simplifier ou compliquer la programmation.

  Carry est annulée par les opérations logiques (AND, OR, XOR). Cela peut être utile car il n'y a pas d'instruction de mise à 0 de Carry, mais seulement de mise à 1 (SCF — *Set Carry Flag*) et d'inversion (CCF — *Complement Carry Flag*).

  Les instructions qui affectent Carry sont : ADC, ADD, SBC, SUB, RL, RR, RLC RRC, RLA, RRA, RLCA, RRCA, SLA, SRA, SRL, DAA, SCF, CCF, AND, OR, XOR

- *L'indicateur N*

  Cet indicateur ne sert pas au programmeur. Il est utilisé de manière interne par l'instruction d'ajustement décimal DAA pour savoir si l'on vient d'effectuer une addition ou une soustraction (l'ajustement est différent dans les deux cas).

  Attention, son nom peut prêter à confusion pour les habitués d'autres microprocesseurs (6502. 6809...) dans lesquels N est l'indicateur de signe (Négatif). Dans le Z80, l'indicateur de signe est S.

□ *L'indicateur P/V* (parité/débordement)

Cet indicateur a également un double rôle : d'autre part, il indique si une opération arithmétique a provoqué un débordement (cf. Chapitre 3). Rappelons qu'un débordement est un changement accidentel de signe du résultat. Dans ce cas, l'indicateur prend le nom de V. V est positionné par toutes les opérations arithmétiques ADC, ADD, SBC, SUB, CP, INC et DEC (sauf doubles registres).

D'autre part, après les opérations logiques, les décalages et rotations ne portant pas sur l'accumulateur, et les instructions DAA, IN r,(C), NEG, il indique si le nombre de bits à 1 dans le résultat est pair. Il prend alors le nom de P (parité). P = 1 indique un nombre pair de bits à 1, et bien sûr P=0 un nombre impair de bits à 1.

De plus, lors des instructions de traitement de blocs (LDD, LDI, LDDR, LDIR, CPD, CPI, CPDR, CPIR), c'est lui qui indique si le registre compteur passe à 0.

□ *L'indicateur H (half carry* — demi-retenue)

Cet indicateur est positionné par une retenue interne du quartet faible vers le quartet fort. Il ne sert pas au programmeur, mais est utilisé par l'instruction DAA lorsqu'on veut travailler en DCB. Dans la pratique, vous n'avez pas à vous en soucier.

□ *L'indicateur* Z (zéro)

L'indicateur Z signale que le résultat d'une opération est nul, ou qu'une comparaison a trouvé l'égalité. Attention, à ce moment Z est mis à 1, il est effacé sinon. Z=0 indique un résultat non nul (risque de confusion !). De même, l'instruction BIT charge dans Z l'inverse du bit testé (Z=1 si bit=0 et réciproquement).

Les instructions de chargement, à l'opposé d'autres microprocesseurs, *n'affectent pas* Z, à l'exception des peu usuels LD A,I et LD A,R. De même, les incrémentations et décrémentations sur doubles registres n'affectent pas Z On devra donc comparer explicitement ces registres avec « 0 » pour savoir s'ils ont été annulés par INC ou DEC.

Les instructions d'entrées-sorties par blocs utilisent Z pour savoir si le compteur (registre B) passe à 0 (INI, IND, OUTI, OUTD, INDR, OTIR. OTDR, INIR).

En plus des précédentes, Z est affecté par ADD (sauf doubles registres), ADC, SUB, SBC, CP, NEG, AND, OR, XOR, ainsi que les décalages et rotations ne portant pas sur l'accumulateur RR, RL, RRC, RLC, RLD, RDD, SLA, SRA, SRL et DAA, IN, BIT, CPI, CPIR, CPD, CPDR.

□ *L'indicateur S (signe)*

Cet indicateur prend la valeur du bit de signe de l'octet qui vient d'être manipulé. Rappelons que le bit de signe est le bit de gauche en représentation en complément à deux (cf. chap. 3 : Arithmétique et logique).

Les instructions de chargement n'affectent pas S, pas plus que INC et DEC doubles registres.

S par contre est affecté par ADD, SUB, SBC, ADC, CP, NEG, AND, OR, XOR, INC, DEC, RR, RL, RRD, RRC, RLC, RLD, SLA, SRA, SRL, DAA, IN, CPR, CPIR, CPD, CPRD, LD A, I, LD A, R.

Bien évidemment, vous ne pouvez pas garder en mémoire tout ce que vous venez de lire. Souvenez vous seulement du rôle de chaque indicateur. Vous pourrez ensuite vous reporter au Jeu d'instructions pour connaître avec précision l'action de telle ou telle instruction sur chacun des indicateurs. De toute manière, il ne sera pas nécessaire dans un programme de connaître à chaque instant la valeur des différents indicateurs, mais seulement en des points particuliers comme par exemple les tests.

LE REGISTRE SP ; CONCEPT DE PILE ; SOUS-PROGRAMMES

■ *La pile*

Une des difficultés rencontrées par le programmeur est de savoir à quelle adresse ranger les données, les résultats ou plus généralement les informations. Il faut en effet que le programme sache où les retrouver plus tard, quand il en aura besoin de nouveau. La **pile** apporte une aide précieuse dans bon nombre de cas.

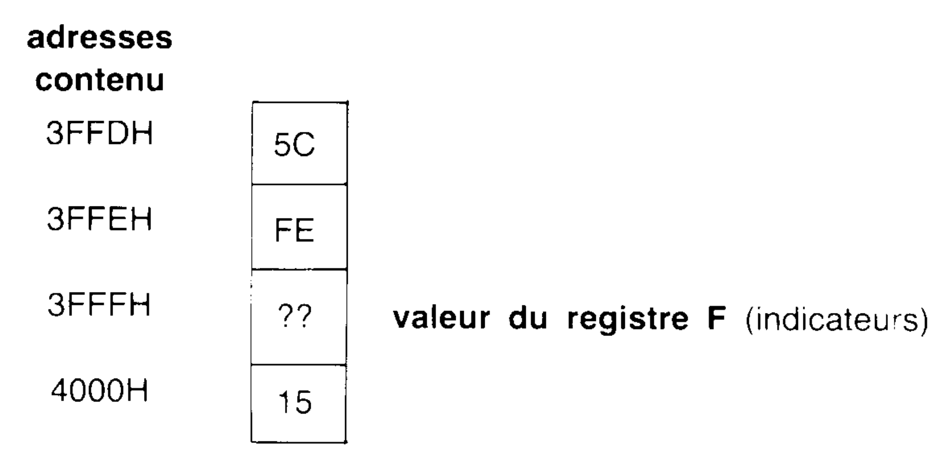
Qu'est-ce que la pile ? C'est une zone de mémoire où l'on « pousse » (PUSH) les informations sans avoir à se soucier d'adresses. Le moment venu, il suffira de les en retirer (POP). La pile fonctionne exactement comme ces piles d'assiettes posées sur un ressort dans les comptoirs des self-services : quand on met des assiettes propres sur le dessus, la pile s'enfonce, et seule la dernière posée affleure ; lorsque vous la prenez, la précédente remonte, et ainsi de suite. Remarquez que la première assiette posée est la dernière à être reprise. C'est ce qu'on appelle une structure LIFO (*Last In, First Out* – dernière entrée, première sortie).

Où la pile se trouve-t-elle ? Là où vous la voulez ! Son adresse est pointée par le registre SP. Avant d'utiliser la pile, il faudra donc charger SP avec l'adresse où vous voulez l'implanter, sachant qu'elle se développera « à l'envers » c'est-à-dire des adresses hautes vers les basses (on arrive à ce paradoxe que le sommet de la pile a une adresse plus basse que sa base !). C'est une des raisons pour lesquelles on représente la mémoire avec les adresses faibles en haut et les adresses fortes vers le bas. En fait cela n'a rien de gênant car la pile fonctionne automatiquement, et si vous avez suffisamment de place mémoire tout ceci reste transparent pour le programmeur.

Exemple :

```
LD SP, 4000H    on implante la pile à l'adresse 4000H
LD A, 15H       on charge 15H dans A
LD HL, 5CFEH    on charge HL avec la valeur 5CFEH
PUSH AF         on empile AF
PUSH HL         on empile HL
```

La pile se présente ainsi :



A chaque empilement, SP est décrémenté du nombre d'octets empilé. Dans l'exemple ci-dessus, SP pointe maintenant sur l'adresse 3FFCH, et le prochain empilement se fera à cette adresse.

A l'inverse, au dépilement, SP sera ré-incrémenté du nombre d'octets dépilés. Notez que les données dépilées ne sont pas effacées du sommet de la pile ; elles y sont lues, puis recopiées dans le registre spécifié. Par contre, le prochain empilement les « écrasera ».

■ *Les sous-programmes*

Qu'est-ce qu'un sous-programme ? C'est un fragment de programme appelé par le programme principal en divers points de son exécution. Vous connaissez déjà cette notion dans le BASIC. Les instructions GOSUB et RETURN vous sont familières. En langage machine, on dispose de la même structure. L'avantage évident est de ne pas avoir à réécrire plusieurs fois une routine dont on aura besoin plusieurs fois dans le courant du programme. L'appel du sous-programme se fait à l'aide de l'instruction CALL. Celle-ci fait deux choses : elle sauvegarde dans la pile le contenu de PC (qui, rappellez-vous, contient l'adresse de la PROCHAINE instruction, donc celle qui suit le CALL), puis met dans PC l'opérande de CALL, c'est-à-dire l'adresse du sous-programme. Celui-ci s'exécute normalement, jusqu'à la rencontre de l'instruction RET (RETurn), qui dépile PC, provoquant le retour au programme principal à l'instruction suivant le CALL. Plusieurs sous-programmes peuvent être imbriqués, le retour se faisant toujours au bon endroit, grâce à la pile. Ceci implique que si vous utilisez la pile dans le courant d'un sous-programme, vous devez TOUJOURS la laisser au moment du retour dans l'état où vous l'avez trouvée, sous peine de ne pas retourner au programme principal, mais, selon toute vraisemblance, de planter la machine ! Il peut être préférable de créer une pile « locale » en sauvegardant SP à une adresse connue et sûre, puis en le chargeant avec l'adresse de la nouvelle pile. On n'oubliera pas de restaurer SP avant la fin de la routine.

5

Les modes d'adressage

Les modes d'adressage sont les différentes manières d'accéder à une donnée. Plus les modes d'adressage sont nombreux, plus le microprocesseur est puissant, et plus grande est sa souplesse de programmation. Le Z80 possède 7 modes d'adressage. Il faut les étudier et les comprendre car leur emploi judicieux est l'une des bases de la programmation, et permet d'alléger considérablement la conception des programmes.

- *L'adressage implicite ou inhérent*

Ce n'est pas vraiment un adressage : les instructions utilisant ce mode « savent » implicitement ce qu'elles doivent faire. Il s'agit d'instructions ayant une action à l'intérieur même du MPU ; les codes opération sont sur un seul octet, et ne demandent pas d'opérande, donc pas de données à aller chercher.

Ce sont : DAA CPL, NEG, SCF, CCF, NOP, HALT, RRCA, RLCA, et EI, DI, IM0, IM1, IM2 qui agissent sur les interruptions.

Exemple : NOP ne fait strictement rien, mais dure 4 cycles !
SCF met la CARRY à 1.

- *L'adressage registre*

Dans ce mode, très utilisé, la donnée se trouve dans un registre (chargé par une opération antérieure). Ce n'est pas non plus un véritable adressage, puisque un registre n'a pas d'adresse, mais un nom. Ici aussi on opère à l'intérieur du MPU ; les codes opération sont le plus souvent sur un seul octet, d'où gain de place et de temps

d'exécution. De nombreuses instructions admettent ce mode qui peut opérer sur des registres 8 bits et 16 bits : LD, EX, EXX, ADD, ADC, SUB, SBC, INC, DEC, AND, OR, XOR, CP, RLC, RL, RR, RRA, RLA, RLC, RRC, SLA, SRA, SRL, IN, OUT, BIT, SET, RES.

Exemple : LD A,B met le contenu de B dans A.
SET 3,C force le bit N° 3 de C à 1.
EX DE,HL échange les contenus DE et HL.

Attention, dans ce mode, comme dans tous les autres, toutes les combinaisons de registres ne sont pas admises avec toutes les instructions. Reportez-vous au chapitre Jeu d'instructions pour avoir tous les détails de telle ou telle instruction.

- *L'adressage immédiat*

Ici, la donnée est spécifiée dans l'instruction, elle suit immédiatement le code opération (d'où le nom d'adressage immédiat). C'est le moyen le plus simple de faire travailler un registre sur une valeur connue.

Faites simplement attention à ne pas essayer d'imposer un nombre 16 bits a un registre 8 bits, ou l'inverse !

Les instructions concernées sont : LD, ADC, ADD, SUB, SBC, AND, OR, XOR, CP.

Voici un exemple qui reprend les modes vus jusqu'ici :

```
LD   A,38H
ADD  A,12H
NEG
INC  A
CP   B
JP   Z,4000H
LD   DE,12CDH
```

La première instruction charge l'accumulateur A avec la valeur 38H, puis ADD lui additionne la valeur 12H, déposant le résultat (4AH) dans A dont l'ancien contenu est perdu. NEG effectue le complément à 2 de A (A devient −A, ici A prend la valeur BBH). INC A incrémente A. A contient maintenant BCH. Ensuite CP B compare le registre B à A (les comparaisons opérant toujours sur A, on ne le spécifie pas). La comparaison est en fait une soustraction ; le résultat n'est pas stocké, mais positionne les indicateurs. JP Z,4000H fera sauter (*JumP* — sauter) l'exécution du programme à l'adresse 4000H si l'indicateur Z est à 1, c'est-à-dire si la comparaison entre A et B a donné l'égalité (A − B = 0). Bien sûr, à l'adresse 4000H doit se trouver un programme commençant par une instruction valide, sous peine de « plantage »... Sinon (A<>B donc Z = 0), l'exécution se poursuit à l'instruction suivante (LD DE,12CDH qui charge la valeur 12CDH (16 bits) dans le double registre DE).

■ *L'adressage direct*

Encore un mode simple : on prend la donnée directement dans l'adresse spécifiée derrière l'instruction. N'oubliez pas : les parenthèses signifient « le contenu de ». Les instructions admettant l'adressage direct sont : LD, JP, CALL, RST, OUT.

Exemple : LD L,(2CDEH) charge le contenu de l'adresse 2CDEH dans le registre L.
JP 5000H saut sans condition à l'adresse 5000H.

■ *L'adressage indirect*

Ce mode est un peu plus compliqué, mais en allant doucement, vous y arriverez ! Jusqu'ici, la donnée était connue au moment de l'écriture du programme. Mais ce n'est pas toujours le cas ; souvent on ne connaît que son adresse, ou un calcul l'aura déposée précédemment. On utilise alors le principe du pointeur que nous avons évoqué au premier chapitre. Ce pointeur sera généralement HL, dans certains cas, BC ou DE. Puisqu'on accède à la donnée indirectement à travers le pointeur, on appelle ce mode l'adressage indirect.

Exemple : LD HL,5400H charge le pointeur HL avec l'adresse connue de la donnée
LD A,(HL) charge A avec la donnée pointée par HL (ici avec le contenu de l'adresse 5400H).

Notez la présence des parenthèses : HL signifie en fait « le contenu de HL » : donc (HL) est bien le contenu du contenu de HL, ici le contenu de 5400H. C'est bien la donnée cherchée, dont on ne connaît pas la valeur.

Il n'existe pas sur le Z80 d'indirection par la mémoire du type LD A,[(adresse)], comme sur d'autres microprocesseurs. C'est quelquefois gênant, mais il faut s'y faire !

Les instructions qui admettent le mode indirect sont : LD, ADD, ADC, SUB, SBC, EX, INC, DEC, AND, OR, XOR, CP, RLC, RRD, RRC, RL, RLD, RR, SLA, SRA, SRL, JP, IN, OUT, BIT, SET, RES, RET, PUSH et POP. Attention, l'indirection ne se fait pas toujours par le même registre (SP pour RET, PUSH et POP par exemple) ; consultez le Jeu d'instructions pour les détails.

■ *L'adressage relatif*

Ce mode intéresse uniquement les instructions de saut. En effet, il y a deux façons d'indiquer l'adresse où l'on veut faire sauter l'exécution : on peut spécifier l'adresse de destination elle-même ; on emploie alors l'instruction JP et le mode direct : JP ADR saute à l'adresse ADR.

Mais on peut aussi indiquer le DEPLACEMENT à ajouter ou à retrancher à l'adresse de l'instruction de saut (en pratique, de celle qui suit le saut) pour trouver la destination. Le saut sera relatif à l'adresse de départ. On emploie par exemple l'instruction JR *(Jump Relative)* qui considère son opérande comme un nombre SIGNE (cf. Chapitre 3 Arithmétique et logique). Ce nombre sera ajouté à PC, provoquant le branchement

désiré. Rappelez-vous qu'un nombre signé est compris entre −128 (0FFH) et +127 (7FH) et que le déplacement possible sera donc de 128 octets en arriere ou 127 octets en avant. L'avantage de cette méthode est de rendre le programme translatable puisqu'on ne spécificie pas d'adresse absolue, et aussi de gagner en place (instruction sur 2 octets au lieu de 3) et en rapidité. Il faut donc l'utiliser chaque fois que possible. Mais rassurez-vous, si vous disposez d'un ASSEMBLEUR symbolique tel que ODIN, il est assez intelligent pour calculer lui-même le déplacement en fonction de l'adresse d'arrivée que vous désirez.

Exemple : JR 0FCH provoque un saut en arrière de 4 octets (0FCH = −4),
JR ARR provoque un saut à l'adresse ARR.

- *L'adressage indirect indexé*

Ce mode reprend le principe de l'adressage indirect, mais en lui ajoutant un déplacement signé. On utilise exclusivement les registres IX et IY. Le déplacement est ajouté au contenu du registre pour donner l'adresse de la donnée. Les instructions qui admettent ce mode sont les mêmes que pour l'adressage indirect par HL, c'est-à-dire toutes celles mentionnées plus haut sauf EX, IN, OUT, RET, PUSH, et PULL.

Exemple :

```
LD      DE,567DH
PUSH    DE
POP     IX
LD      (IX+5),A
```

Il n'existe pas d'instruction LD IX, DE ; si l'on veut faire cette opération, une solution consiste à passer par la pile : après POP IX, IX contient 567DH. LD (IX+5),A met le contenu de A à l'adresse 567DH + 5 = 5682H.

Remarquez que si le déplacement est nul, on revient à l'adressage indirect par IX ou IY. Cependant, il vaut mieux employer HL quand c'est possible, car les codes opération avec IX+d ou IY+d sont toujours plus longs (en place et en temps), même avec d nul, qu'avec HL.

Nous avons fait le tour des modes d'adressage présents sur le Z80. Précisons toutefois que certaines instructions utilisent plusieurs modes en même temps (ce qui ne simplifie pas la compréhension... !). Par exemple LD (IX+0AH), 0B3H charge la valeur immédiate 0B3 dans l'adresse pointée par IX+0AH. L'adressage est immédiat, indirect et indexé ! Le plus souvent, ce sont les instructions les plus puissantes qui combinent d'elles-mêmes les différents modes : DJNZ, LDI, LDIR, LDD, LDDR, CPI, CPD, CPIR, CPDR, INI, OUTI, IND, OUTD, OTIR, INIR, OTDR, INDR (cf. Jeu d'instructions).

Il faut bien comprendre que c'est la syntaxe même des mnémoniques qui indique quel(s) mode(s) est (sont) utilisé(s).

Exercice 5.1 : Voici une suite d'instructions ; décrivez le fonctionnement de chacune d'elles, et dites quel est le contenu des registres et ou cases mémoire concernées (elles ont toutes été expliquées plus haut, alors, si vous avez des difficultés, relisez tout ce chapitre pour l'assimiler).

```
    LD   D,90H
    LD   E,0
    EX   DE,HL
    LD   (HL),03
    LD   IX,9005H
ICI LD   B,(IX-5)
    LD   A,07
    CP   B
    JR   NZ,ICI
    PUSH HL
    POP  BC
```

6

Le jeu d'instructions du Z80

Ce chapitre contient 5 grands groupes d'instructions dont voici le détail :

— Groupe 1 : Transferts de données
— Groupe 2 : Opérations arithmétiques
— Groupe 3 : Opérations logiques, décalages, rotations
— Groupe 4 : Branchements, sauts, tests, opérations sur la pile
— Groupe 5 : Interruptions, entrées sorties, divers.

Conventions d'écriture :

Le jeu d'instructions du Z80 est très étendu puisqu'il comporte près de 700 codes opération différents ! Pour simplifier la lecture (et diminuer l'épaisseur de ce livre !) nous avons regroupé les instructions semblables. Les conventions suivantes vous diront comment lire ces instructions ; il s'agit de conventions internes à cet ouvrage et, en écrivant vos programmes, vous devez bien sûr expliciter toutes les opérandes à chaque instruction.

— Les registres seront appelés par la (ou les) lettre(s) qui leur correspond(ent), et que vous connaissez déjà.
— Les indicateurs du registre F seront appelés par la lettre qui leur correspond.
Pour indiquer l'action des instructions sur ces indicateurs, nous prendrons les conventions suivantes :

0 si l'indicateur est mis à 0
1 si l'indicateur est mis à 1
* si l'indicateur est modifié en fonction du résultat de l'opération
? si l'indicateur est modifié de manière aléatoire.

— r désignera un registre 8 bits (A, B, C, D, E, H, L).
— dr désignera un double registre (donc 16 bits), soit BC, DE, HL, SP, IX ou IY. Certaines instructions ne portent que sur quelques-uns de ces doubles registres. C'est pourquoi nous indiquerons à chaque fois quel(s) double(s) registre(s) dr peut représenter.
— Ind désignera un registre d'index (IX ou IY).
— data8 désignera une donnée sur 8 bits.
— data16 désignera une donnée sur 16 bits.
— ad désignera une adresse (16 bits).

— op désignera un opérande variable. Nous indiquerons à chaque fois ce qu'il peut représenter.
— cond désignera une condition sur les indicateurs. Exemple, JR NC,8000H provoque le saut à l'adresse 8000h si, et seulement si, l'indicateur C est à 0. Les conditions seront explicitées à chaque fois.
— Les parenthèses () désigneront le contenu de l'emplacement mémoire qu'elles encadrent. S'il s'agit du contenu d'un registre, nous omettrons ces parenthèses afin d'alléger l'écriture et de conserver la cohérence avec les mnémoniques. En revanche, comme nous l'avons déjà vu, (dr) signifiera « le contenu du contenu du double registre dr », c'est-à-dire le contenu de l'adresse pointée par dr.
— d désignera un deplacement, soit un nombre 8 bits que le Z80 considère comme SIGNE (cf. Chapitre 2).
— Le symbole > désignera le ET logique.
— Le symbole > désignera le OU logique.
— Le symbole A désignera le OU exclusif (XOR).
— Le symbole « : = » sera celui de l'affectation ; cela consiste à calculer la valeur qui se trouve à droite de ce signe et à l'affecter à la variable qui se trouve à sa gauche. Ainsi la « phrase » A : = A + 1 signifiera prendre le contenu du registre A, y ajouter 1, puis affecter cette nouvelle valeur au registre A.
— Les différents modes d'adressage ne seront pas expliqués à chaque fois. C'est inutile car vous savez maintenant que les parenthèses désignent l'adressage indirect, que le nom d'un registre désigne l'adressage registre, qu'une instruction sans opérande a un adressage inhérent, et que data8 ou data16 indique l'adressage immédiat. Si des doutes subsistent, consultez l'Annexe 5, ou toutes les instructions sont reprises avec leur code opération, leur temps d'exécution, et leurs modes d'adressage.

Groupe 1
Les transferts de données

Remarques générales sur les instructions de chargement :

- Les indicateurs ne sont pas modifiés (sauf par LDA,I et LD A R, peu usuels).
- Si le point de départ d'un chargement est le contenu d'un registre ou d'une case mémoire, celui-ci n'est pas modifié (il est simplement lu).
- L'ancien contenu du registre ou de la case mémoire chargé par une instruction est perdu.
- Pour des raisons de commodité et de place, nous avons regroupé le plus possible les instructions similaires. Nous avons toujours indiqué sur quel(s) registre(s) elles agissent ; attention, toutes les combinaisons ne sont pas toujours possibles ; en cas d'incertitude, reportez vous à l'Annexe 5, où vous trouverez une liste complète du Jeu d'instructions, classé par ordre alphabétique.
- Le nombre d'octets que comportent les instructions de chargement va de 1 à 4 ; reportez-vous à l'annexe 5 pour connaître le nombre d'octets de l'instruction qui vous intéresse.
- Le déplacement dans le mode indexé est normalement fixé à l'écriture du programme. C'est un nombre signé (−128<déplacement< +127) qui est codé dans le dernier octet de l'instruction. Une astuce pour faire varier ce déplacement consiste à faire varier cet octet :

```
        LD  A,03
        LD  LABEL + 2,A
LABEL   LD  D,(IX + 5)
```

  chargera en fait D avec le contenu de IX+3. Cette méthode permet d'obtenir un adressage indexé avec déplacement calculé, et non pas fixe.
- En adressage direct, du type LD op, (ad) ou LD (ad), op, l'adresse ad est codée sur les deux derniers octets de l'instruction, poids faible d'abord. Une astuce du même type que ci-dessus est possible pour modifier cette adresse dans le courant du programme.

LD r,op

(Chargement d'un registre avec une valeur)

Effet : r := op
r est l'un des registres A, B, C, D, E, H, L.
op peut être r, data8, (HL), (IX+d), (IY+d)

Description : Le registre spécifié est chargé avec l'opérande mentionnée

Indicateurs : SZ H PNC
aucun effet

Exemples :

avant		après
B:56H ; A:?	LD A,B	B:56H ; A:56H
C:?	LD A,0FCH	C:0FCH
IX:5000H		IX:5000H
(5006H):28H		(5000H):28H
L:?	LD L,(IX+6)	L:28H

Remarque : LD A,A, LD B,B... sont autorisés (mais à quoi servent-ils ?).

LD A,op

(Chargement de l'accumulateur avec une valeur)

Effet : A := op
op peut être A, B, C, D, E, H, L, (BC),(DE), (HL), (IX+D), (IY+D), (ad), data8.

Description : L'accumulateur est chargé avec l'opérande spécifiée.

Indicateurs : SZ H PNC
aucun effet

Exemple :

avant		après
(2035H):02H		(2035H):02H
A:?	LD A,2035H	A:02H

Remarque : Cette instruction englobe la précédente dans le cas où r est A. A est « privilégié » par rapport aux autres registres dans la mesure où il admet l'indirection par BC, DE, ou une adresse quelconque ad.

LD op,A

(Stockage de l'accumulateur dans un registre ou un emplacement mémoire)

Effet : op := A
op peut être A, B, C, D, E, H, L, (BC), (DE), (HL), (IX + d), (IY + d), (ad).

Description : Le contenu de l'accumulateur est chargé dans l'opérande spécifiée.

Indicateurs : SZ H PNC
aucun effet

Exemple :

avant		après
BC:6000		BC:6000
A:0FFH		A:0FFH
(6000):?	LD (BC),A	(6000):0FFH

Remarque : Même remarque que pour LD A,op.

LD dr,(ad)

(Chargement d'un double registre avec le contenu des adresses ad et ad +1)

Effet : partie basse de dr := (ad)
partie haute de dr := (ad + 1)
dr peut être BC, DE, HL, IX, IY ou SP.

Description : Le contenu de l'adresse ad est chargé dans la partie basse du double registre spécifié ; le contenu de l'adresse ad + 1 est chargé dans sa partie haute.

Indicateurs : SZ H PNC
aucun effet

Exemple :

avant	LD DE,(5402H)	après
(5403H):4CH		(5403H):4CH
(5402H):0AAH		(5402H):0AAH
DE:?		DE:4CAAH

LD dr, data16

(Chargement d'un double registre avec une donnée immédiate 16 bits.)

Effet : dr := data16
dr peut être BC, DE, HL, IX; IY ou SP.

Description : La valeur 16 bits data16 est chargée dans le double registre spécifié.

Indicateurs : SZ H PNC
aucun effet

Exemple :	**avant**	LD IX,67FEH	**après**
	IX:?		IX:67FEH

Remarque : Utilisée avec SP, cette instruction change la pile de place; c'est utile... ou dangereux (retours de sous-programmes)!

LD (ad),dr

(Stockage d'un double registre aux adresses ad et ad + 1)

Effet : (ad) := partie basse de dr
(ad + 1) := partie haute de dr

dr peut être BC, DE, HL, IX, IY ou SP

Description : L'adresse ad est chargée avec l'octet le moins significatif de dr; l'adresse ad + 1 est chargée avec l'octet le plus significatif de dr.

Indicateurs : SZ H PNC
aucun effet

Exemple :	**avant**	**après**
	(FF30H):?	(FF30H):8CH
	(FF31H):?	(FF31H):01
	HL 018CH	HL:018CH

LD (dr),op

(Stockage d'un registre ou d'une valeur à l'adresse pointée par un double registre)

Effet : (dr) := op
dr peut être HL, IX+d, IY+d
op peut être A, B, C, D, E, H, L ou data8

Description : L'adresse pointée par le double registre mentionné est chargée avec le contenu du registre spécifié ou avec la donnée immédiate indiquée.

Indicateurs : SZ H PNC
aucun effet

Exemples :

avant		après
IX:7089H (7089H):? C:12H	LD (IX),C	IX:7089H (7089H):12H C:12H
IY:5000 (5007):? E:0	LD (IY + 7),E	IY:5000 (5007):00 E:0
HL:5566H (5566H):?	LD (HL),H	HL:5566H (5566H):55H

Dans ce dernier exemple, puisque HL contient 5566H, cela signifie que H contient 55H, et que L contient 66H.

Remarque : L'indirection de B, C, D, E, H et L ne peut se faire qu'au travers de HL ou d'un registre d'index (avec ou sans déplacement). L'indirection au travers de BC ou DE ne peut se faire qu'avec A. L'instruction LD (BC), D par exemple, bien que concevable, n'existe pas sur le Z80.

LD A,I ou R

(Chargement de l accumulateur à partir du registre I ou du registre R)

Effet : A := I
ou A := R

Description : L'accumulateur est chargé avec le contenu du registre de vectorisation des interruptions (I) ou avec le contenu du registre de rafraîchissement dynamique des mémoires (R).

Indicateurs : SZ H PNC
** 0 ? 0

Remarque : Vous n'utiliserez pas LD A,I - I ne servant pas sur les machines MSX. LD A,R vous donnera un nombre pseudo-aléatoire, ce qui peut avoir son utilité.

LD R ou I,A

(Chargement de R ou de I à partir de A)

Effet : R:= A
ou I := A

Indicateurs : SZ H PNC
aucun effet

Remarque : Vous n'avez pas à vous servir de ces instructions très techniques.

LD SP,dr

(Chargement du pointeur de pile à partir d'un double registre)

Effet : SP := dr
dr peut être HL, IX, IY.

Description : Le registre pointeur de pile SP est chargé avec le contenu du double registre spécifié.

Indicateurs : SZ H PNC
aucun effet

Exemple :	**avant**		**après**
	SP:?		SP:8000H
	IX:8000H	LD SP,IX	IX:8000H

Remarque : SP ne peut être chargé qu'à partir de HL, IX, IY. Voir aussi LD dr,(ad) ; LD dr, data16 ; LD (ad),dr.
De toute manière, soyez très prudent en ce qui concerne la manipulation de la pile (cf. chap. 3, Sous-programmes).

LDD

(Transfert de bloc avec auto-décrémentation)

Effet : (DE) := (HL). DE := DE − 1; BC := BC − 1 ; HL := HL − 1.

Description : Le contenu de l'adresse pointée par HL est chargé dans l'adresse pointée par DE, puis BC, DE, HL sont décrémentés tous les trois.

Indicateurs : SZ H PNC
0 * 0
P = 1 si BC = 0
P = 0 si BC<>0

Exemple :

avant	LDD	après
BC:02A5H		BC:02A4H
DE:5200H		DE:51FFH
HL:876DH		HL:876CH
(5200H):?		(5200H):37H
(876DH):37H		(876DH):37H

Remarque : Dans les 4 instructions LDD, LDI, LDDR, LDIR, la 3e lettre indique si on a affaire à une Incrémentation (I) ou à une Décrémentation (D).
BC sert de compteur de boucle, c'est pourquoi il est décrémenté (ou incrémenté dans le cas de LDI) automatiquement ; HL doit au préalable être chargé avec l'adresse du premier octet à transférer, et DE avec son adresse de destination. DE et HL sont ensuite préparés pour le transfert suivant.

LDDR

(Transfert de bloc avec décrémentation et répétition automatique)

Effet : (DE) := (HL) ; DE := DE − 1 ; BC:=BC − 1 ; HL := HL − 1 ; répétition jusqu'à ce que BC = 0.

Description : Le contenu de l'adresse pointée par HL est chargé dans l'adresse pointée par DE, puis BC, DE, HL sont décrémentés tous les trois. SI BC n'a pas été annulé par cette décrémentation, on recommence (PC est décrémenté de 2 car LDDR a 2 octets de long).

Indicateurs : SZ H PNC
0 00

Remarque : BC sert de compteur ; il doit être chargé avec le nombre d'octets à transférer ; HL doit être chargé avec l'adresse du premier octet à transférer, et DE avec son adresse de destination. Cette instruction est très puissante. Elle permet d'effectuer des transferts de blocs (groupes d'octets) très rapidement.

LDI

(Transfert de bloc avec auto-incrémentation)

Effet : (DE) := (HL) ; DE := DE + 1 ; HL := HL + 1 ; BC := BC − 1.

Description : Identique à LDD, mais ici HL et DE sont incrémentés.

Indicateurs : SZ H PNC
comme LDD

Remarque : Voir LDD.

LDIR

(Transfert de bloc avec incrémentation et répétition automatique)

Effet : (DE) := (HL) ; DE := DE + 1 ; BC := BC − 1 ; HL := HL + 1 ; répétition jusqu'à ce que BC = 0.

Description : Le contenu de l'adresse pointée par HL est chargé dans l'adresse pointée par DE, puis DE, HL sont incrémentés, et BC est décrémenté. Si BC n'a pas été annulé par cette incrémentation, on recommence (PC est décrémenté de 2 car LDDR a 2 octets de long).

Indicateurs : SZ H PNC
0 00

Remarque : BC sert de compteur : il doit être chargé avec le nombre de transferts à opérer : HL doit être chargé avec l'adresse du premier octet à transférer, et DE avec son adresse de destination. Cette instruction est très puissante. Elle permet d'effectuer des transferts de blocs (groupes d'octets) très rapidement.

Comment savoir s'il faut employer LDDR ou LDIR (ou LDD LDI) ? Cela dépend : si la zone d'arrivée ne recouvre pas la zone de départ, pas de problème, LDDR et LDIR sont équivalentes. Par contre, s'il y a recouvrement, il faut considérer la direction du transfert ; si l'on transfère vers le haut de la mémoire, il faut employer LDDR de manière à éviter d'écraser les donnees à transférer. De cette façon, quand, après plusieurs décrémentations, on commence à recouvrir la zone de départ, les données recouvertes ont déjà été transférées, et on ne perd pas d'information. Evidemment, si l'on transfère vers le bas de la mémoire, c'est l'inverse, et il faut employer LDIR. Ces remarques sont aussi valables pour LDD et LDI.

EX DE,HL

(Echange des registres HL et DE)

Effet : DE ⟷ HL

Description : Le contenu du double registre DE est échangé avec celui de HL.

Indicateurs : SZ H PNC
aucun effet

Exemple :

	avant	EX DE,HL	**après**
	DE : 46CBH		DE : 67F3H
	HL : 67F3H		HL : 46CBH

Remarque : Les indicateurs ne sont pas positionnés par cette instruction. Attention donc, si vous voulez savoir si HL a été annulé, il faudra faire le test explicitement (cf. DEC dr ou INC dr).

EX (SP),HL ou Ind

(Echange de HL ou de Ind avec le sommet de la pile)

Effet : L ⟷ (SP) ; H ⟷ (SP+1)
ou ind $_{bas}$ ⟷ (SP) ; Ind$_{haut}$ ⟷ (SP+1)

Description : Le contenu du registre L ou de la partie basse du registre index est échangé avec celui de l'adresse pointée par SP ; le contenu de H ou de la partie haute du registre d'index est échangé avec celui de l'adresse immédiatement supérieure.

Indicateurs : SZ H PNC
aucun effet

Exemple : SP : 6000H

avant	EX (SP),HL	**après**
HL : 3A45H		HL : 0BC8FH
(6001H) : 0BCH		(6001H) : 3AH
(6000H) : 0C8H		(6000H) : 45H

Rermarque : Les instructions EX (SP),HL, EX (SP),IX, EX (SP),IY fonctionnent de manière identique ; c'est pourquoi nous les avons regroupées. Attention, après l'exécution de l'instruction EX (SP),HL, le prochain dépilement affectera l'ancienne valeur de HL au registre dépilé. Cela peut être utile, mais gare aux retours de sous-programmes si vous avez modifié la pile ! (Un sous-programme doit toujours rendre la pile dans l'état où il l'a trouvée en entrant, à moins que vous ne soyez très fort...)

EX AF, AF'

(Echange de l'accumulateur et du registre d'état avec leurs homologues de la 2e banque de registres)

Effet : AF ←→ AF'

Description : Le contenu de l'accumulateur et les indicateurs sont échangés avec ceux du groupe auxiliaire de registres.

Indicateurs : SZ H PNC
aucun effet

Remarque : Le registre F étant échangé, les indicateurs prennent les valeurs qui étaient dans F' ; celles qui étaient dans F vont dans F'.

EXX

(Echange des banques de registres)

Effet : BC ←→ BC' ; DE ←→ DE' ; HL ←→ HL'

Description : Les contenus de BC, DE et HL sont échangés avec les contenus des registres auxiliaires correspondants.

Indicateurs : SZ H PNC
aucun effet

Remarque : EXX est avec EX AF, AF' la seule instruction agissant sur les 2e banques de registres. En fait, EXX agit comme une bascule : on utilise une banque ou l'autre, jamais les deux à la fois. Cela permet de stocker 6 octets et de les récupérer ensemble avec une seule instruction.

Groupe 2
Opérations arithmétiques

ADC A,op

(Addition de l'accumulateur, de l'opérande et de Carry)

Effet : A : = A + op + C
op peut représenter : A, B, C, D, E, H, L, data8, (HL), (IX+d), (IY+d).

Description : L'opérande ainsi que la Carry C sont additionnées à l'accumulateur. Le résultat est placé dans l'accumulateur. Attention, cette opération faisant intervenir la Carry C, vous devez donc en connaître le contenu auparavant car le résultat pourrait être différent de celui attendu.

Indicateurs : SZ H VNC
** * *0 *

Exemple : **avant**	ADD A,C	**après**
A : 2AH		A : 5DH
C : 33H		C : 33H
Carry : 0		Carry : 0

ADC HL,dr

(Addition de HL, d'un double registre et de Carry)

Effet : HL : = HL + dr + C
dr peut être BC, DE, HL ou SP.

Description : Le contenu du registre HL et celui du double registre spécifié sont additionnés avec la Carry C. Le résultat est placé dans le registre double HL. Les mêmes précautions qu'avec l'instruction précédente doivent être prises quant à la Carry C. A noter que cette instruction est avec ADD HL,dr la seule à effectuer des additions sur 16 bits.

Indicateurs : SZ H VNC
** ? *0*

H est positionné en fonction du report sur le bit 11.

Exemple : **avant** ADC HL,BC **après**

avant		après
HL : A35EH		HL : B5FEH
BC : 12A0H		BC : 12A0H
Carry : 0		Carry : 0

Remarque : L'instruction ADC HL,HL peut permettre de multiplier HL par 2 de manière élégante (la Carry doit bien entendu être mise à 0 auparavant par exemple à l'aide de OR A).

ADD A,op

(Addition de l'accumulateur et de l'opérande)

Effet : A : = A + op
op peut représenter A, B, C, D, E, H, L, data8, (HL), (IX+d), (IY+d).

Description : Le contenu de l'opérande est additionné (sans retenue, à la différence de l'instruction ADC A,op) au contenu de l'accumulateur. Le résultat est placé dans l'accumulateur. Etant donné que l'opération est effectuée sans retenue, vous n'avez pas à vous occuper de l'état de celle-ci avant d'exécuter cette instruction.

Indicateurs : SZ H VNC
** * *0*

Exemple :	**avant**	ADD A,2AH	**après**
	A : E4H		A : 1EH
	Carry : ?		Carry : 1

Remarque : Il peut vous paraître bizarre que la Carry passe à 1 lors de l'exécution de cette instruction. Pourtant, ceci n'a rien de contradictoire avec ce que nous venons de dire : en effet, l'addition est effectuée sans prise en compte de l'état de la Carry. Cependant, si cette addition engendre une retenue, la Carry est alors mise à 1. Ceci explique pourquoi ici la Carry est positionnée à 1.

ADD HL,dr

(Addition de HL avec le double registre dr)

Effet : HL : = HL + dr
dr peut être BC, DE, HL, ou SP.

Description : Le contenu du double registre HL est additionné au contenu du double registre dr. Le résultat est placé dans HL. L'addition est effectuée sans prise en compte de l'état de la Carry.

Indicateurs :
```
SZ H VNC
   ?  0*
```

H est positionné s'il y a report du bit 11.
C est positionné s'il y a report du bit 15.

Exemple :

avant	ADD HL,HL	**après**
HL : 5046H		HL : A08CH
		Carry : 0

(cette opération multiplie HL par 2)

Remarque : Seuls C et H sont affectés. S, Z et P ne sont pas modifiés.

ADD Ind,dr

(Addition d'un index avec le double registre dr)

Effet : Ind : = Ind + dr
Ind peut être IX ou IY
dr peut être BC, DE, SP ou Ind

Description : Le contenu du registre d'index est additionné au contenu du double registre dr. Le résultat est placé dans le registre d'index. L'addition est effectuée sans prise en compte de l'état de la Carry.

Indicateurs :
```
SZ H VNC
   ?  0*
```

H est positionné en fonction du report sur le bit 11.
C est positionné en fonction du report sur le bit 15.

Exemple : avant ADD IX,BC **après**

avant		après
IX : A790H	ADD IX,BC	IX : 7BE0H
BC : D450H		BC : D450H
		Carry : 1

Remarque : On ne peut pas, avec cette fonction, additionner IX à IY (ou IY à IX). On peut additionner seulement IX à IX ou IY à IY.
Seuls C et H sont affectés. S,Z et P ne sont pas modifiés.

SBC A,op

(Soustraction de l'opérande et de Carry à l'accumulateur)

Effet : A : = A − op − C
op peut représenter : A, B, C, D, E, H, L, data8, (HL), (IX+d), (IY+d).

Description : Le contenu de l'opérande est dans un premier temps ajouté au contenu de la Carry, puis l'ensemble est soustrait du contenu de l'accumulateur. Le résultat est placé dans l'accumulateur. Il s'agit d'une opération faisant intervenir la Carry, vous devez donc bien en connaître le contenu au préalable.

Indicateurs : SZ H VNC
** * *1*

Exemple : avant SBC A,D **après**

avant		après
A : 3FH	SBC A,D	A : 2CH
D : 12H		D : 12H
Carry : 1		Carry : 0

SBC HL,dr

(Soustraction du double registre dr et de la Carry à HL)

Effet : HL : = HL − dr C
dr peut être BC, DE, HL, ou SP.

Description : Le contenu du registre double dr est, dans un premier temps, ajouté au contenu de la Carry, puis l'ensemble est soustrait au contenu du double registre HL. Le résultat est placé dans HL. Les mêmes précautions qu'à l'instruction précédente sont à prendre en ce qui concerne la Carry.

Indicateurs : SZ H VNC
** ? *1 *

H est positionné s'il y a report du bit 12.

Exemple :

avant	SBC HL,BC	**après**
HL : 3F3EH		HL : 0534H
BC : 3A0AH		BC : 3A0AH
Carry 0		Carry : 0

Remarque : Si la Carry est à 0, un SBC HL,HL annule le registre double HL. Ceci peut être utile car l'instruction SBC HL,HL n'occupe que deux octets alors que l'instruction LD HL,0 en nécessite trois En revanche, l'instruction SBC HL,HL nécessite 15 cycles d'horloge alors que LD HL,0 n'en a besoin que de 10. Le gain de place se fait donc au détriment d'une perte de temps.

SUB op

(Soustraction de l'opérande à l'accumulateur)

Effet : A : = A − op
op peut représenter A, B, C, D, E, H, L, data8, (HL), (IX+d), (IY+d).

Description : Le contenu de l'opérande est soustrait du contenu de l'accumulateur. Le résultat est placé dans l'accumulateur.
Contrairement à l'instruction SBC A,op, on n'a pas à se préoccuper de l'état de la Carry avant d'effectuer cette instruction.

Indicateurs : SZ H VNC
* * * * 1 *

Exemple :

avant	SUB A,(HL)	**après**
A : A5H		A : 52H
HL : 3F40H		HL : 3F40H
(3F40H) : 53H		(3F40H) : 53H

Remarque : Bien que certains ASSEMBLEURS admettent SUB A, op (cohérent avec SBC A, op), la mnémonique officielle est bien SUB op ; A étant sous-entendu comme dans CP op. L'exécution de SUB A annule l'accumulateur en 4 cycles et en 1 octet alors que LD A,0 l'annule en 7 cycles et 2 octets. La première façon est donc plus économique aussi bien au point de vue temps que place.

INC op

(Incrémentation de l'opérande spécifié)

Effet : op : = op + 1
op peut représenter A, B, C, D, E, H, L, (HL), (IX+d), (IY+d)

Description : Cette instruction ajoute 1 au contenu de l'opérande spécifiée. Le résultat est replacé dans l'opérande.

Indicateurs : SZ H VNC
** * *0

Exemple : avant	INC B	**après**
B : 27H		B : 28H

INC dr

(Incrémentation du double registre dr)

Effet : dr : = dr + 1
dr peut etre BC, DE, HL, ou SP.

Description : Le double registre dr spécifié est incrémenté. Le résultat est placé dans ce même double registre.

Indicateurs : SZ H VNC
aucun effet

Exemple : avant	INC BC	**après**
BC : 3F5FH		BC : 3F60H

Remarque : Vous devez garder présent à l'esprit que l'incrémentation d'un double registre (de même que sa décrémentation) n'affecte pas le registre d'état. De ce fait, le passage à zéro ne sera pas signalé par le positionnement de l'indicateur Z. Vous devez donc vous-même tester la valeur du registre double pour savoir s'il y a eu ou non passage à zéro. Pour cela, il faut d'abord tester la partie basse, puis, si elle est nulle, la partie haute.

Exemple pour voir si DE est nul :

```
            LD  A,E
            OR  D         ;tous les bits de E et de D nuls ?
            JP  Z,NUL     ;si oui DE=0
  PASNUL    ---           ;sinon DE< >0
```

Si le contenu de A ne doit pas être perdu, ajoutez PUSH AF au début, et POP AF aux adresses NUL et PASNUL pour le sauvegarder dans la pile.

INC Ind

(Incrémentation du registre d'index Ind)

Effet : Ind : = Ind + 1

Description : Le registre d'index spécifié est incrémenté. Le résultat est placé dans ce même registre d'index.

Indicateurs : SZ H VNC
aucun effet

Exemple : avant	INC IX	**après**
IX : 3F45H		IX : 3F46H

Remarque : Les indicateurs n'étant pas modifiés par l'exécution de cette instruction, méfiez-vous lors de son exécution, en particulier si le passage à zéro vous importe, vous devez faire le test vous-même, ce qui n'est pas très simple ; il faut faire par exemple :

```
PUSH  IX
POP   DE   ;charge DE avec le contenu de IX
```

suivis de la routine de la page précédente. Si, de plus, DE doit être conservé, utilisez EXX et la 2e banque de registres.

DEC op

(Décrémentation de l'opérande spécifiée)

Effet : op : = op − 1
op peut être r, (HL), (IX+d), (IY+d)

Description : Le contenu de l'opérande est diminué de 1. Le résultat est placé dans cette même opérande.

Indicateurs : SZ H VNC
** * *1

Exemple : avant	DEC (HL)	après
HL : 3456H		HL : 3456H
(3456) : F7H		(3456H) : F8H

DEC dr

(Décrémentation du double registre dr)

Effet : dr : = dr − 1
dr peut être BC, DE, HL ou SP.

Description : Le contenu de dr est décrémenté. Le résultat est placé à nouveau dans dr.

Indicateurs : SZ H VNC
aucun effet

Exemple : avant	DEC DE	après
DE : 35A0H		DE : 359FH

Remarque : De même que l'instruction INC dr, cette instruction ne modifie pas le contenu du registre d'état. Vous êtes donc invités à faire les tests vous-même si vous en avez besoin (cf. Remarque pour INC dr).

DEC Ind

(Décrémentation du registre d'index Ind)

Effet : Ind : = Ind − 1

Description : Le contenu du registre d'index spécifié est décrémenté. Le résultat est à nouveau stocké dans ce même registre d'index.

Indicateurs : SZ H VNC
aucun effet

Exemple : avant	DEC IY	**après**
IY : 0000H		IY : FFFFH

Remarque : L'exécution de cette instruction ne modifie pas le contenu du registre d'état. Vous devez donc faire vous-même explicitement les tests de passage à zéro si vous en avez besoin (cf. Remarque pour INC Ind).

CPL

(Complémentation à 1 de l'accumulateur)

Effet : Ind : A : = $\overline{A}$

Description : Le contenu de l'accumulateur est complémenté à 1. Le résultat est placé dans l'accumulateur. La complémentation signifie que l'on remplace chaque bit à 0 par un 1 et chaque bit à 1 par un 0.

Indicateurs : SZ H VNC
1 1

Exemple : avant	CPL	**après**
A : 6DH		A : 92H
(01101101)		(10010010)

NEG

(Opposé de l'accumulateur)

Effet : A : = – A

Description : On soustrait l'accumulateur de la valeur 0 et on stocke le résultat dans l'accumulateur. Cette opération s'appelle une complémentation à 2.

Indicateurs : SZ H PNC
** * * * *
C est positionné si A valait 0.
P est positionné si A valait 80H.

Exemple :	**avant**	NEG	**après**
	A : ACH		A : 54H

Remarque : Le complement à 2 est en fait égal au complément à 1 augmenté de 1.

DAA

(Ajustement décimal de l'accumulateur)

Effet : A : = A ajusté décimal

Description : L'accumulateur est ajusté en fonction du contenu du registre d'état. Un nombre sur 8 bits (placé dans A) peut être décomposé en deux quartets : un quartet de poids fort Qh (quartet haut) et un quartet de poids faible Qb (quartet bas). Si l'on désire travailler en DCB (voir chapitre Arithmétique et logique), chacun de ces quartets doit contenir un nombre compris entre 0 et 9 (les valeurs A à F sont à proscrire). Cependant, le Z80 ne sait faire que des opérations arithmétiques binaires. En DCB, il faut donc, après une opération arithmétique, effectuer une correction sur le résultat ; c'est ce que se charge de faire cette instruction DAA.

Indicateurs : SZ H PNC
** * * *

Exemple :	**avant**	DAA	**après**
	A : 4CH		A : 52H

Groupe 3
Opérations logiques, décalages rotations ; tests de bits

AND op

(Et logique entre l'accumulateur et l'opérande op)

Effet : A : = A ∧ op
op peut être A, B, C, D, E, H, L, data8, (HL), (IX+d), (IY+d).

Description : Le ET logique entre l'accumulateur et l'opérande spécifiée est effectué. Le résultat est rangé dans l'accumulateur.

Indicateurs : SZ H PNC
** 1 *00

Exemple : **avant**	AND 08H	**après**
A : 32H		A : 00

Remarque : A n'apparaît pas dans la mnémonique ; c'est inutile car cette instruction agit toujours sur lui, et il n'y a pas d'ambiguïté.

AND A a pour unique effet de mettre Carry à 0. C'est intéressant car il n'y a pas d'instruction spécifique pour cela ; la séquence SCF CCF (voir ces instructions) le fait aussi, mais en deux octets contre un.

OR op

(OU logique entre l'accumulateur et l'opérande op)

Effet : A : = A V op
op peut être A, B, C, D, E, H, L, data8, (HL), (IX+d), (IY+d).

Description : Le OU logique entre l'accumulateur et l'opérande spécifiée est effectué. Le résultat est rangé dans l'accumulateur.

Indicateurs : SZ H PNC
** 0 *00

Exemple :

avant	OR (HL)	après
A : 22H		A : 33H
HL : 6904H		HL : 6904H
(6904H) : 11H		(6904H) : 11H

Remarque : A n'apparaît pas dans la mnémonique ; c'est inutile car cette instruction agit toujours sur lui, et il n'y a pas d'ambiguïté.

OR A a pour unique effet de mettre Carry à 0. C'est intéressant car il n'y a pas d'instruction spécifique pour cela ; la séquence SCF CCF (voir ces instructions) le fait aussi, mais en deux octets contre un.

XOR op

(OU EXCLUSIF logique entre l'accumulateur et l'opérande op)

Effet : A : = A ∀ op
op peut être A, B, C, D, E, H, L, data8, (HL), (IX+d), (IY+d).

Description : Le OU EXCLUSIF (voir chapitre 2) logique entre l'accumulateur et l'opérande spécifiée est effectué. Le résultat est rangé dans l'accumulateur.

Indicateurs : SZ H PNC
⋆⋆ 0 ⋆00

Exemple : **avant** **après**

avant		après
A : 22H	XOR B	A : 03H
B : 21H		B : 21H

Remarque : A n'apparaît pas dans la mnémonique ; c'est inutile car cette instruction agit toujours sur lui, et il n'y a pas d'ambiguïté.

XOR A a pour effet d'annuler l'accumulateur (en un octet) et de mettre Carry à 0. C'est intéressant car il n'y a pas d'instruction spécifique pour cela ; la séquence SCF CCF (voir ces instructions) le fait aussi, mais en deux octets contre un.

SCF

(Mise à 1 de Carry (Set Carry Flag)*)*

Effet : C : = 1

Description : Carry est mise à 1.

Indicateurs : SZ H PNC
0 01

Remarque : Utilisée par exemple avant une rotation.

CCF

(Complémentation de Carry)

Effet : C : = $\overline{C}$

Description : Inverse l'état de Carry.

Indicateurs : SZ H PNC
? 0*

Remarque : Il n'y a pas d'instruction de mise à 0 de Carry, mais vous pouvez le faire facilement avec XOR A (qui annule aussi l'accumulateur), OR A ou AND A, ou encore CP A qui met aussi Z à 1.

BIT b,op

(Test du bit b de op)

Effet : Z : = inverse du bit No b de op
op peut être A, B, C, D, E, H, L, (HL), (IX+d), (IY+d).

Description : Le bit b de l'opérande spécifiée est testé ; s'il est à 1, l'indicateur Z est mis à 0 ; s'il est à 0, Z est mis à 1.

Indicateurs : SZ H PNC
?* 1 ?0

Exemple :

avant	BIT 2,H	**après**
H : 01110001 (= 71H)		H : 01110001 (= 71H)
Z :?		Z : 1

Remarque : N'oubliez pas que les bits d'un octet sont numérotés de 0 à 7 à partir de la droite. Cette instruction très pratique est souvent utilisée après une lecture d'un port d'entrée. Elle est généralement suivie d'un saut conditionnel sur Z.

SET b,op

(Mise à 1 du bit b de op)

Effet : bit N° b de op : = 1
op peut être A, B, C, D, E, H, L, (HL), (IX+d), (IY+d).

Description : Le bit b de l'opérande spécifiée est mis à 1.

Indicateurs : SZ H PNC
aucun effet

Exemple :

avant		après
D : 00000000	SET 5,D	D : 00100000

Remarque : Si le bit en question est déjà à 1, cette instruction n'a pas d'effet.

RES b,op

(Mise à 0 du bit b de op)

Effet : Z : bit N° b de op : = 0
op peut être A, B, C, D, E, H, L, (HL), (IX+d), (IY+d).

Description : Le bit b de l'opérande spécifiée est mis à 0.

Indicateurs : SZ H PNC
aucun effet

Exemple :

avant		après
A : 11101101	RES 0,A	A : 11101100

Remarque : Si le bit en question est déjà à 0, cette instruction n'a pas d'effet.

RL op

(Rotation à gauche de op à travers Carry)

Effet :

```
 ┌──────────────────────┐
 └─[C] ← [7 ← 0]←┘
            op
```

op peut être A, B, C, D, E, H, L, (HL), (IX+d), (IY+d).

Description : Le contenu de l'opérande spécifiée est décalé d'un bit vers la gauche ; le bit 0 prend la valeur de l'indicateur C ; l'indicateur C prend la valeur du bit 7. Le résultat est remis à l'emplacement d'origine. La rotation se fait donc sur 9 bits.

Indicateurs : SZ H PNC
** 0 *0*

Exemple : avant		après
L:01100111 (=47H)	RL L	L:11001110 (=CEH)
C:0		C:0

Remarque : Si Carry est à 0 avant, cette opération revient à multiplier op par 2 (vérifiez-le à titre d'exercice).

RLA

(Rotation à gauche de l'accumulateur à travers Carry)

Effet :

```
 ┌──────────────────────┐
 └─[C] ← [7 ← 0] ←──────┘
            A
```

Description : Le contenu de l'accumulateur est décalé d'un bit vers la gauche ; le bit 0 prend la valeur de l'indicateur C ; l'indicateur C prend la valeur du bit 7. Le résultat est remis dans l'accumulateur. La rotation se fait donc sur 9 bits.

Indicateurs : SZ H PNC
  0 0 *

Exemple : avant **après**

avant		après
A:11010001	RLA	A:10100010
(D1H)		(A2H)
C:0		C:1

Remarque : Vous pouvez aussi obtenir le même effet avec l'instruction précédente en faisant RL A. La différence est que RLA ne prend qu'un octet contre 2 pour RL A, et qu'elle est donc plus rapide (adressage inhérent contre adressage registre). En contrepartie, RLA ne positionne ni S, ni Z, ni P. Si C = 0 avant, RLA multiplie A par 2.

RR op

(Rotation à droite de op à travers Carry)

Effet :

→ [7 → 0] → [C] —

op

op peut être A, B, C, D, E, H, L, (HL), ((IX+d), (IY+d)).

Description : Le contenu de l'opérande spécifiée est décalé d'un bit vers la droite ; le bit 7 prend la valeur de l'indicateur C ; l'indicateur C prend la valeur du bit 0. Le résultat est remis à l'emplacement d'origine. La rotation se fait donc sur 9 bits.

Indicateurs : SZ H PNC
** 0 *0*

Exemple : **avant** RL (IX+2) **après**

avant		après
IX:CBDCH		IX:CBDCH
(CBDEH):01110010		(CBDEH):10111001
(=72H)		(=B9H)
C:1		C:0

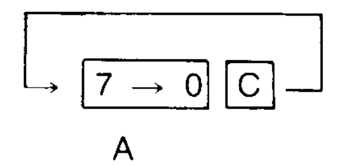
Remarque : Si C = 0 avant, il s'agit d'une division par 2 de op, le reste étant dans Carry (vérifiez-le à titre d'exercice).

RRA

(Rotation à droite de l'accumulateur à travers Carry)

Effet :





Description : Le contenu de l'accumulateur est décalé d'un bit vers la droite ; le bit 7 prend la valeur de l'indicateur C ; l'indicateur C prend la valeur du bit 0. Le résultat est remis dans l'accumulateur. La rotation se fait donc sur 9 bits.

Indicateurs : SZ H PNC
0 0 *

Exemple :

avant		**après**
A:11010001	RRA	A:01101000
(=D1H)		(=68H)
C:0		C:1

Remarque : Vous pouvez aussi obtenir le même effet avec l'instruction précédente en faisant RR A. La différence est que RRA ne prend qu'un octet contre 2 pour RR A, et qu'elle est donc plus rapide (adressage inhérent contre adressage registre). En contrepartie, RRA ne positionne ni S, ni Z, ni P. Si C = 0 avant, RRA divise A par 2, reste dans Carry.

RLC op

(Rotation à gauche de op)

Effet :

[C] ← [7 ← 0] ← (bit 7 renvoyé vers bit 0)

op

op peut être A, B, C, D, E, H, L, (HL), (IX+d), (IY+d).

Description : Le contenu de l'opérande spécifiée est décalé circulairement d'un bit vers la gauche ; le bit 7 va en même temps dans le bit 0 et dans l'indicateur C. Le résultat est remis à l'emplacement d'origine. La rotation se fait donc sur 8 bits.

Indicateurs : SZ H PNC
** 0 *0*

Exemple : avant	RLC H	**après**
H:00111011 (=3BH)		L:01110110 (=76H)
C:0		C:0

Remarque : Attention aux mnémoniques ! Le C de RLC signifie SANS Carry. Il y a là un risque d'erreur possible ; l'inverse aurait été plus logique, mais adressez-vous à ZILOG...

RLCA

(Rotation à gauche de l'accumulateur)

Effet :

```
        ┌──────────────┐
[C] ←─┘ [7 ← 0] ←─┘
            A
```

Description : Le contenu de l'accumulateur est décalé d'un bit vers la gauche ; le bit 7 va en même temps dans le bit 0 et dans l'indicateur C. Le résultat est remis dans l'accumulateur. La rotation se fait donc sur 8 bits.

Indicateurs : SZ H PNC
0 0 *

Exemple :	**avant**		**après**
	A:01110101	RLCA	A:11101010
	(=75H)		(=EAH)
	C:0		C:0

Remarque : Gardez à l'esprit que le C de RLCA signifie SANS Carry. Vous pouvez aussi obtenir le même effet avec l'instruction précédente en faisant RLC A. La différence est que RLC A ne prend qu'un octet contre 2 pour RLC A, et qu'elle est donc plus rapide (adressage inhérent contre adressage registre). En contrepartie, RLCA ne positionne ni S, ni Z, ni P.

RRC op

(Rotation à droite de op)

Effet :

7 → 0 C

op

op peut être A, B, C, D, E, H, L, (HL), (IX+d),(IY+d).

Description : Le contenu de l'opérande spécifié est décalé circulairement d'un bit vers la droite ; le bit 0 va en même temps dans le bit 7 et dans l'indicateur C. Le résultat est remis à l'emplacement d'origine. La rotation se fait donc sur 8 bits.

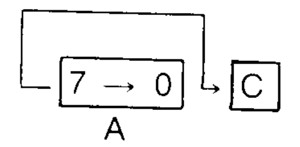
Indicateurs : SZ H PNC
* * 0 *0*

Exemple : avant		**après**
B:10111011 (=BBH)	RRC B	B:11011101 (=DDH)
C:?		C:1

RRCA

(Rotation à droite de l'accumulateur)

Effet :



Description : Le contenu de l'accumulateur est décalé d'un bit vers la droite ; le bit 0 va en même temps dans le bit 7 et dans l'indicateur C. Le résultat est remis dans l'accumulateur. La rotation se fait donc sur 8 bits.

Indicateurs : SZ H PNC
0 0*

Exemple : avant		**après**
A:01110101 (=75H)	RRCA	A:10111010 (=BAH)
C:?		C:1

Remarque : Attention, le C de RRCA signifie SANS Carry. Vous pouvez aussi obtenir le même effet avec l'instruction précédente en faisant RRC A. La différence est que RRCA ne prend qu'un octet contre 2 pour RRC A, et qu'elle est donc plus rapide (adressage inhérent contre adressage registre). En contrepartie, RRCA ne positionne ni S, ni Z, ni P.

SLA op

(Décalage arithmétique à gauche de op)

Effet :

[C] ← [7 ← 0] ← 0

op

op peut être A, B, C, D, E, H, L, (HL), (IX+d), (IY+d)

Description : Le contenu de l'emplacement spécifié est décalé à gauche d'un bit. Le bit 7 tombe dans l'indicateur C, et il entre un 0 par la droite dans le bit 0. Le résultat est remis à l'emplacement de départ.

Indicateurs : SZ H PNC
** 0 *0*

Exemple : avant		**après**
L:10010011	SLA L	L:00100110
(93H)		(26H)
C:?		C:1

Remarque : Bien que la lettre A figure dans la mnémonique (SLA — *Shift Left Arithmetical*, décalage arithmétique à gauche), cette instruction peut aussi agir sur les autres registres ou en mode indirect.
SLA effectue en fait la multiplication par 2 de op.

SRA op

(Décalage arithmétique à droite de op)

Effet :

```
┌→ |7 → 0| → |C|
└──┘  op
```

op peut être A, B, C, D, E, H, L, (HL), (IX+d), (IY+d).

Description : Le contenu de l'emplacement spécifié est décalé à droite d'un bit. Le bit 0 tombe dans l'indicateur C, et le bit 7 est inchangé. Le résultat est remis à l'emplacement de départ.

Indicateurs : SZ H PNC
* * 0 * 0 *

Exemple :

avant		après
IY:BC12H	SRA (IY)	IY:BC12H
(BC12H):11001011		(BC12H):11100101
C:?		C:1

Remarque : Bien que la lettre A figure dans la mnémonique (SRA — *Shift Right Arithmetical*, décalage arithmétique à droite), cette instruction peut aussi agir sur les autres registres ou en mode indirect.

SRL op

(Décalage logique à droite de op)

Effet :

0 → [7 → 0] → [C]

op

op peut être A, B, C, D, E, H, L, (HL), (IX+d), (IY+d).

Description : Le contenu de l'emplacement spécifié est décalé à droite d'un bit. Le bit 0 tombe dans l'indicateur C, et il entre un 0 à gauche dans le bit 7. Le résultat est remis à l'emplacement de départ.

Indicateurs : SZ H PNC
★★ 0 ★0★

Exemple :

avant		après
HL:A0A0H	SRL (HL)	HL:A0A0H
(A0A0H):11100001		(A0A0H):01110000
C:?		C:1

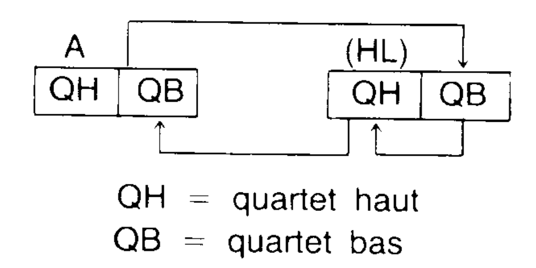
Remarque : SRL effectue en fait la division par 2 de op, le reste étant déposé dans Carry.

Toutes ces instructions de décalage se ressemblent un peu. N'hésitez pas à revenir consulter ces pages à chaque fois que vous en avez besoin.

RLD

(Rotation décimale à gauche)

Effet :



QH = quartet haut
QB = quartet bas

Description : Rotation par quartets entre (HL) et l'accumulateur. Le quartet faible du contenu de l'adresse pointée par HL est décalé dans le quartet fort de cette même adresse. Le quartet fort de (HL) va dans le quartet faible de A ; le quartet faible de A va dans le quartet faible de (HL).

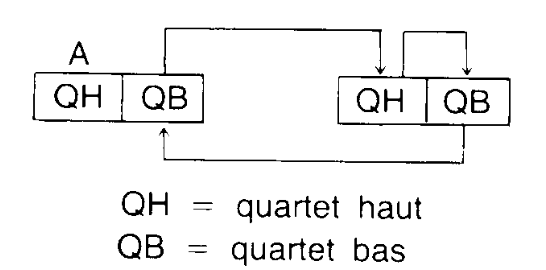
Indicateurs : SZ H PNC
** 0 *0

Remarque : Cette instruction est utilisée pour faciliter les calculs DCB.

RRD

(Rotation décimale à droite)

Effet :



QH = quartet haut
QB = quartet bas

Description : Rotation par quartets entre (HL) et l'accumulateur. Le quartet fort du contenu de l'adresse pointée par HL est décalé dans le quartet faible de cette même adresse. Le quartet faible de (HL) va dans le quartet faible de A ; le quartet faible de A va dans le quartet faible de (HL).

Indicateurs : SZ H PNC
** 0 *0

Remarque : Cette instruction est utilisée pour faciliter les calculs DCB.

Groupe 4
Comparaisons, sauts, pile, sous-programmes

CP op

(Comparaison entre l'accumulateur et l'opérande op)

Effet : A – op
op peut être A, B, C, D, E, H, L, data8, (HL), (IX+d), (IY+d).

Description : L'opérande spécifiée est soustraite de l'accumulateur. Le résultat n'est pas conservé, mais positionne les indicateurs.

Indicateurs : SZ H VNC
★★ ★ ★1★

Exemple : **avant**		**après**
A:32H	CP 08H	SZ H PNC 00 1 010

Remarque : A n'apparaît pas dans la mnémonique ; c'est inutile, car cette instruction agit toujours sur lui, et il n'y a pas d'ambiguité.

S est positionné par le bit de signe du résultat.

Z est mis à 1 si op = A, mis à 0 sinon.

C est mis à 1 si A < op, mis à 0 si A >= op (ceci est valable pour des valeurs positives ; si vous travaillez en arithmétique signée, c'est un peu plus compliqué, car il faut tenir compte de V. En pratique, considérez que les comparaisons opèrent sur des nombres non signés et intégrez le bit de signe dans la valeur de l'octet pour votre raisonnement).

CPD

(Comparaison et décrémentation)

Effet : A – (HL) HL := HL–1 ; BC := BC–1.

Description : Le contenu de l'adresse pointée par HL est soustrait de l'accumulateur. Le résultat n'est pas conservé, mais affecte les indicateurs. Ensuite, HL et BC sont tous deux décrémentés (et prêts pour une nouvelle comparaison).

Indicateurs :

```
SZ H PVC
*|  *  1
 |    └→mis à 0 si BC a
 |      été annulé, mis à 1
 |      sinon.
 └────→ positionné si A=(HL)
```

Remarque : Ici, c'est P qui indique que BC est passé par 0 (dans d'autres instructions ce peut être Z), et Z qui indique l'égalité dans la comparaison. BC sert de compteur d'octets c'est pourquoi il est décrémenté automatiquement ; HL doit au préalable être chargé avec l'adresse du 1er octet à comparer.

CPDR

(Comparaison, décrémentation, et répétition)

Effet : A – (HL) ; HL := HL–1 ; BC := BC–1 : si A <> (HL) et BC < > 0, PC := PC–2 (réexécution).

Description : Le contenu de l'adresse pointée par HL est soustrait à l'accumulateur. Le résultat n'est pas gardé, mais positionne les indicateurs. Ensuite, HL et BC sont décrémentés. L'instruction se réexécute jusqu'à ce que la comparaison réussisse ou que BC = 0.

Indicateurs :

```
SZ H PVC
*|  *  |1
 |     └→mis à 0 si BC a
 |       été annulé, mis à 1
 |       sinon.
 └────→  positionné si A=(HL)
```

Remarque : Instruction puissante qui peut par exemple chercher automatiquement un octet donné dans une table. HL doit pointer sur le début de la table (côté des adresses hautes), et BC doit contenir la longueur de cette table. En sortie, si la table contient la valeur cherchée, HL contient l'adresse de la 1ère occurrence à partir des adresses hautes, sinon BC contient 0.

CPI

(Comparaison et décrémentation)

Effet : A − (HL) ; HL := HL+1 ; BC := BC−1.

Description : Le contenu de l'adresse pointée par HL est soustrait de l'accumulateur. Le résultat n'est pas conservé, mais affecte les indicateurs. Ensuite, HL est incrémenté, et BC est décrémenté (et ils sont prêts pour une nouvelle comparaison).

Indicateurs : SZ H PVC
* * (Z) : positionné si A=(HL)
* * (H)
* 1 (N)
* P/V : mis à 0 si BC a été annulé, mis à 1 sinon.

Remarque : Voir CPD.

CPIR

(Comparaison, incrémentation et répétition)

Effet : A − (HL) ; HL := HL+1 ; BC := BC−1 ; si A ≠ (HL) et BC ≠ 0, PC := PC−2 (réexécution).

Description : Le contenu de l'adresse pointée par HL est soustrait à l'accumulateur ; le résultat n'est pas gardé, mais positionne les indicateurs. Ensuite, HL est incrémenté, et BC est décrémenté. L'instruction se réexécute jusqu'à ce que la comparaison réussisse ou que BC = 0.

Indicateurs : SZ H PVC
* * (Z) : positionné si A=(HL)
* * (H)
* 1 (N)
* P/V : mis à 0 si BC a été annulé, mis à 1 sinon.

Remarque : Instruction puissante qui peut par exemple chercher automatiquement un octet donné dans une table. HL doit pointer sur le début de la table (côté des adresses basses), et BC doit contenir la longueur de cette table. En sortie, si la table contient la valeur cherchée, HL contient l'adresse de la 1ère occurrence à partir des adresses basses, sinon BC contient 0.

JP ad

(Saut inconditionnel à l'adresse ad)

Effet : PC := ad

Description : La valeur ad est chargée dans PC, provoquant le saut à l'adresse ad.

Indicateurs : SZ H PNC
aucun effet

Remarque : Cette instruction, contrairement à CALL ad, ne permet pas de retour. Elle est utilisée par exemple pour sauter une zone mémoire contenant des données ou simplement pour brancher dans une autre partie du programme.

JP cond,ad

(Saut conditionnel à l'adresse ad)

Effet : si cond vraie, comme JP ad.
si cond fausse, pas d'effet.
cond peut être Z (Z=1)
NZ (Z=0)
C (C=1)
NC (C=0)
PO (P=1, parité paire)
PE (P=0, parité impaire)
M (S=1, négatif)
P (S=0, positif)

Description : L'indicateur spécifié est testé. Si la condition est remplie, on saute à l'adresse ad. Si la condition n'est pas remplie, on passe à l'instruction suivante (soit PC := PC+3).

Indicateurs : SZ H PNC
aucun effet

Exemples : a) si l'indicateur C=1, JP C,6789H effectue le saut
b) si l'indicateur C=0, JP C,F56CH n'effectue pas le saut
c) si l'indicateur Z=0, JP NZ,6664H effectue le saut

Remarque : C'est l'une des instructions les plus importantes. Le test des indicateurs est la base de la programmation ; il permet d'appliquer tel ou tel traitement suivant les résultats du calcul précédent. Cf. CP op pour le sens des indicateurs après une comparaison.

JP (dr)

(Saut inconditionnel à l'adresse contenue dans le double registre dr)

Effet : PC := dr
dr peut être HL, IX ou IY.

Description : Le contenu du double registre dr est chargé dans PC, provoquant le saut à cette adresse.

Indicateurs : SZ H PNC
aucun effet

Exemple : HL: 7080H
JP (HL) provoque le saut à l'adresse 7080H.

Remarque : Attention, les parenthèses dans la mnémonique ont été voulues par ZILOG, mais ne sont pas logiques car il ne s'agit pas d'une indirection : on saute à l'adresse contenue dans dr, et non à celle contenue dans l'adresse pointée par dr.

Cette instruction permet le saut à une adresse calculée, ce que ne font pas les autres instructions de saut.

JR d

(Saut relatif de déplacement d)

Effet : PC := PC+d

Description : Le deplacement fourni d est ajouté au compteur ordinal PC, provoquant un saut de d octets. d est un nombre signé ; le déplacement possible est donc de −128 octets en arrière à +127 octets en avant, comptés à partir de l'adresse de l'instruction qui suit le JR (puisque PC contient l'adresse de la prochaine instruction).

Indicateurs : SZ H PNC
aucun effet

Exemple : JR 0F0H effectue un saut en arrière de 16 octets (F0H = −16 en complément à 2)

Remarque : Si vous possédez ODIN de Loriciels, ou un autre ASSEMBLEUR symbolique, il calculera d pour vous ; vous n'aurez qu'à indiquer l'adresse absolue de branchement de manière explicite ou symbolique.

JR cond,d

(Saut relatif conditionnel de déplacement d)

Effet : si cond vraie, comme JR d.
si cond fausse, pas d'effet.
cond peut être Z (Z=1)
NZ (Z=0)
C (C=1)
NC (C=0)

Description : L'indicateur spécifié est testé. Si la condition est remplie, on effectue le saut relatif d comme décrit pour JR d. Si la condition n'est pas remplie, on passe à l'instruction suivante (PC := PC+2).

Exemples : — si l'indicateur Z=1, JR Z,67H effectue un saut en avant de 103 octets en avant (comptés à partir de l'instruction suivante)
— si l'indicateur C=0, JR C,9 est ignoré.

Remarque : Cette instruction prend 2 octets, alors que JP cond,d en prend 3. De plus, le saut relatif permet un relogement facile du programme en cas de besoin. Cependant, malgré ces avantages, elle ne peut tester que 2 indicateurs au lieu de 4 pour JP. Elle ne peut donc pas toujours remplacer celle-ci. Voir aussi remarque de JR ad.

DJNZ d

(Décrémentation de B et saut relatif si B <>0)

Effet : B := B−1 ; si B<>0, PC := PC+d

Description : B est décrémenté ; si cette opération ne l'a pas annulé, on opère le saut relatif de déplacement d. d est un nombre signé. Le déplacement possible est donc de −128 octets en arrière à +127 octets en avant, comptés à partir de l'adresse de l'instruction suivant le DJNZ.

Indicateurs : SZ H PNC
aucun effet

Remarque : Instruction intéressante qui permet de faire facilement des boucles. Le registre B sert de compteur et doit être chargé avec le nombre de boucles à exécuter. Si vous possédez ODIN de Loriciels, ou un autre ASSEMBLEUR symbolique, il calculera d pour vous ; vous n'aurez qu'à indiquer l'adresse absolue de branchement de manière explicite ou symbolique.

PUSH dr

(Empilement du double registre dr)

Effet : SP := SP-1 ; (SP) := partie forte de dr
SP := SP-1 ; (SP) := partie faible de dr
dr peut être AF, BC, DE, HL, IX, IY

Description : SP est décrémenté ; la partie forte de dr est chargée à la nouvelle adresse pointée par SP. SP est de nouveau décrémenté, et la partie basse de dr est chargée à l'adresse pointée par SP. A la fin de l'opération, SP pointe donc sur la partie basse de la valeur empilée, et SP+1 pointe sur sa partie haute.

Indicateurs : SZ H PNC
aucun effet

Remarque : La pile croît à l'envers : chaque nouvel empilement se place SOUS le précédent, c'est-à-dire aux deux adresses immédiatement inférieures (c'est pour remédier à ce « paradoxe » qu'on représente généralement la mémoire avec les adresses fortes en bas).

POP dr

(Dépilement d'un double registre)

Effet : partie faible de dr := (SP) ; SP := SP+1 ;
partie forte de dr := (SP) ; SP := SP+1
dr peut être AF, BC, DE, HL, IX ou IY.

Description : Le contenu de l'adresse pointée par SP est chargé dans la partie faible du double registre spécifié ; SP est incrémenté. Le contenu de la nouvelle adresse pointée par SP est chargée dans la partie forte du double registre spécifié. SP est de nouveau incrémenté pour pointer sur la prochaine valeur à dépiler.

Indicateurs : SZ H PNC
aucun effet

Remarque : Les instructions de chargement direct entre doubles registres sont quasi inexistantes sur le Z80, à l'exception de EX, DE, HL et de LD SP,dr. On peut cependant les simuler à l'aide de PUSH et POP :

```
PUSH  IX
POP   BC
```

équivaut à LD BC,IX qui n'existe pas. Vous pouvez ainsi fabriquer toutes les combinaisons entre doubles registres.

CALL ad

(Appel de sous-programme)

Effet : d'abord PUSH PC, puis PC := ad.

Description : PC est empilé comme si l'on exécutait PUSH PC. Ensuite PC est chargé avec la valeur ad, provoquant le branchement à l'adresse ad. Le retour se fera à la rencontre de l'instruction RET, qui dépilera PC.

Indicateurs : SZ H PNC
aucun effet

Remarque : PC contient à chaque instant l'adresse de la PROCHAINE instruction à exécuter. C'est donc l'adresse de l'instruction qui suit le CALL qui sera empilée, et le programme reprendra bien après lui au retour du sous-programme

Veillez à TOUJOURS laisser la pile dans le même état aussi bien à la fin d'un sous-programme qu'à son début. Cela signifie que dans le courant d'un sous-programme, il doit y avoir le même nombre de PUSH que de POP, sous peine de ne pas retourner au programme principal. Seule exception : vous voulez retourner ailleurs qu'après le CALL dans le programme principal. Mais cette technique est délicate, et est déconseillée aux débutants !

CALL cond,ad

(Appel conditionnel de sous-programme)

Effet : si cond vraie, comme CALL ad.
si cond fausse, pas d'effet.
cond peut être Z (Z=1)
NZ (Z=0)
C (C=1)
NC (C=0)
PO (P=1, parité paire)
PE (P=0, parité impaire)
M (S=1, négatif)
P (S=0, positif)

Description : L'indicateur spécifié est testé. Si la condition est remplie, on appelle le sous-programme d'adresse ad. Si la condition n'est pas remplie, on passe à l'instruction suivante (PC := PC+3)

Exemples : — si l'indicateur Z=0, CALL Z,ad n'effectue pas le branchement
— si l'indicateur C=1 CALL C,ad effectue le branchement
— si l'indicateur S=0, CALL M,ad n'effectue pas le branchement.

Remarque : Cette instruction est l'une des plus importantes et des plus utiles. Elle permet souvent, en fonction de résultats de calculs qui affectent les indicateurs, une comparaison d'appliquer tel ou tel traitement, puis de revenir au programme principal (RET). cf. CP op pour le sens des indicateurs après une comparaison.

RET

(Retour de sous-programme)

Effet : PC := (SP)

Description : PC est dépilé comme si l'on exécutait POP PC. Ceci provoque le retour au programme principal, à condition que la pile n'ait pas été modifiée.

Indicateurs : SZ H PNC
aucun effet

Remarque : On peut imbriquer de nombreux sous-programmes les uns dans les autres. Chaque RET renvoie juste après le CALL qui l'a appelé (il doit toujours y avoir le même nombre de PUSH que le POP à l'intérieur d'un sous-programme). Cette structure est identique à celle du GOSUB... RETURN du BASIC.

RET cond

(Retour conditionnel de sous-programme)

Effet : si cond vraie, comme RET
si cond fausse, ignoré
cond peut être Z (Z=1)
NZ (Z=0)
C (C=1)
NC (C=0)
PO (P=1, parité paire)
PE (P=0, parité paire)
M (S=1, négatif)
P (S=0, positif)

Description : L'indicateur spécifié est testé. Si la condition est vérifiée, PC est dépilé, provoquant le retour du sous-programme. Si la condition n'est pas remplie, on continue en séquence (l'instruction suivante est exécutée).

Indicateurs : SZ H PNC
aucun effet

Exemple : Si l'indicateur Z=1, RET Z provoque le retour de sous-programme.

Remarque : Voir RET et CP op pour le sens des indicateurs après une comparaison.

RST ad

(Redépart à l'adresse ad)

Effet : PC est empilé comme PUSH PC ; PC := ad
ad peut être 00, 08H, 10H, 18H, 20H, 28H, 30H, 38H.

Description : Après avoir empilé PC, on saute à l'adresse page 0 spécifiée.

Indicateurs : SZ H PNC
aucun effet

Exemple : RST 28H

Remarque : Cette instruction ne prend qu'un octet, elle est donc rapide. Elle donne accès à 8 routines de la ROM fréquemment appelées par l'interpréteur BASIC dont nous donnerons les détails plus loin.

RETI et RETN

(Retour de sous-programme d'interruption)

Effet : Comme RET mais pas utilisé sur MSX.

Groupe 5
Entrées/sorties, interruptions, divers

Un certain nombre d'instructions du Z80 ne sont pas utilisées sur le système MSX. Nous les citerons, mais ne les décrirons pas.

De plus, le fonctionnement en entrée/sortie, décrit plus loin, est particulier à chaque système. Tout ce qui sera dit dans les pages sur les instructions d'entrées/sorties sera spécifique au MSX. Le fonctionnement général du Z80 en interruption et en entrée/sortie est plus complexe. Si vous voulez des détails, consultez une documentation spécialisée sur le Z80.

IN r,(C)

(Chargement d'un registre à partir du port adressé par C)

Effet : r := (C)
r peut être A, B, C, D, E, H, ou L.

Description : Le port d'entrée dont l'adresse est dans le registre C est lu, et le résultat est mis dans l'accumulateur.

Indicateurs : SZ H PNC
* * * * 0

Exemple : **avant** IN A,(C) **après**

avant		après
C:A8H		C:A8H
port A8H:03H		port A8H:03H
A:?		A:03H

Remarque : Seuls quelques ports sont utilisés dans la configuration standard MSX (cf. carte Entrées/Sorties) et leur adresse est toujours inférieure à 255 (FFH). C'est pourquoi elle tient sur 8 bits dans le registre C.

IN A,(P)

(Chargement de l'accumulateur à partir du port P)

Effet : A := (port P)
P est un nombre entre 0 et FFH (8 bits).

Description : Le port d'adresse P est lu ; le résultat est mis dans l'accumulateur.

Indicateurs : SZ H PNC
aucun effet

Exemple :	**avant**		**après**
	A:?	IN A,(79H)	A:F3H
	port 79H:F3H		port 79H:F3H

Remarque : IN A,(P) n'a pas d'effet sur les indicateurs alors que IN r,(C) en a. Les parenthèses sont logiques : c'est bien le contenu du port No P qui est lu. Seuls quelques ports sont utilisés sur MSX (cf. carte Entrées Sorties).

IND

(Entrée et décrémentation)

Effet : (HL) := (C) ; B := B−1 ; HL := HL−1.

Description : Le port adressé par le registre C est lu ; le résultat est placé à l'adresse pointée par HL. Ensuite le registre B et le double registre HL sont décrémentés (pour préparer une prochaine entrée).

Indicateurs : SZ H PNC
?* ? ?1
└── mis à 1 si B passe à 0.

Exemple :	**avant**	IND	**après**
	HL:7050H		HL:704FH
	B:5H		B:4H
	(7050H):?		(7050H):A7H
	C:A0H		C:A0H
	port A0H:A7H		port A0H:A7H

Remarque : Le registre B sert de compteur d'entrées. Cette instruction permet de transférer en mémoire une suite de B lectures du port adressé par le registre C, à l'aide d'une boucle.

INI

(Entrée avec incrémentation)

Effet : (HL) := (C) ; B := B−1 ; HL := HL+1.

Description : Idem que pour IND en remplaçant « HL est décrémenté » par « HL est incrémenté ».

Indicateurs : SZ H PNC
?* ? ?1

↑___ mis à 1 si B passe à 0, mis à 0 sinon.

Remarque : Idem que pour IND.

INDR et INIR

(Entrées de bloc avec décrémentation ou incrémentation et répétition automatique)

Description : Ces instructions agissent exactement comme IND et INI, avec une répétition automatique si B n'a pas été annulé.

Indicateurs : SZ H PNC
?1 ? ?1

Remarque : Certains périphériques demandent un petit laps de temps entre deux lectures ; il faut alors utiliser une boucle associée à INI, IND ou simplement IN.

OUT (C),r

(Sortie d'un registre vers le port adressé par C)

Effet : port (C) :- r
r peut être A, B, C, D, E, H, ou L.

Description : Le contenu du registre r est écrit dans le port dont l'adresse est dans C.

Indicateurs : SZ H PNC
aucun effet

Exemple :

avant		après
H:08H	OUT (C),H	H:08H
C:97H		C:97H
port 97H:?		port 97H:08H

Remarque : Seuls quelques ports sont actifs (voir carte Entrées Sorties). Leurs adresses sont toujours inférieures à FFH, donc tiennent sur 8 bits dans le registre C.

OUT (P),A

(Sortie de l'accumulateur sur le port P)

Effet : (port P) := A

Description : Le contenu de l'accumulateur est écrit dans le port d'adresse P. P est un nombre entre 0 et 255 (FFH). En fait, seuls quelques ports sont actifs sur MSX (voir carte Entrées/Sorties).

Indicateurs : SZ H PNC
aucun effet

Exemple : **avant** OUT (A9H),A **après**

avant		après
(port A9H):?	OUT (A9H),A	(port A9H):05
A:05		A:05

OUTD

(Sortie et décrémentation)

Effet : (C) := (HL) ; B := B−1 ; HL := HL−1.

Description : Le contenu de l'adresse pointée par le double registre HL est écrit sur le port adressé par le registre C. Ensuite, le registre B et le double registre HL sont décrémentés (pour préparer une prochaine sortie).

Indicateurs : SZ H PNC
?* ? ?1
↑__mis à 1 si B passe à 0.

Remarque : B sert de compteur (8 bits). Cette instruction peut servir à écrire un bloc de données dans un périphérique (écran ou imprimante par exemple).

OUTI

(Sortie avec incrémentation)

Effet : (C) := (HL) ; B := B−1 ; HL := HL+1.

Description : Le contenu de l'adresse pointée par le double registre HL est écrit sur le port adressé par le registre C. Ensuite le registre B est décrémenté et le double registre HL est décrémenté (pour préparer une prochaine sortie).

Indicateurs : SZ H PNC
?* ? ?1
└ mis à 1 si B passe à 0.

Remarque : B sert de compteur (8 bits). Cette instruction peut servir à écrire un bloc de données dans un périphérique (écran ou imprimante par exemple).

OTIR et OTDR

(Sorties par bloc avec incrémentation ou décrémentation et repétition automatique)

Description : Ces instructions agissent exactement comme OUTI et OUTD avec une répétition automatique si B n'a pas été annulé.

Indicateurs : SZ H PNC
?1 ? ?1

Remarque : Certains périphériques demandent un petit temps entre deux écritures ; il faut alors utiliser une boucle associée à OUTI, OTDR ou simplement OUT.

IM0,IM1,IM2

(Sélection du mode de fonctionnement en interruption)

Description : Sur le système MSX, c'est le mode IM1 qui est utilisé. Vous ne devez pas changer ce mode, sous peine de « planter la machine ». Ne vous servez donc pas de ces instructions, citées pour mémoire.

DI

(*Interdiction des interruptions* (Disable Interrupts))

Effect : Les interruptions sont interdites, jusqu'à la rencontre de l'instruction EI.

Indicateurs : SZ H PNC
aucun effet

Remarque : Lorsque les interruptions sont interdites, vous gagnez en vitesse de calcul, mais aucun périphérique ne peut plus intervenir. Il faut donc toujours les autoriser à la fin de votre routine, pour reprendre le contrôle de l'ordinateur.

EI

(*Autorisation des interruptions* (Enable Interrupts))

Effet : Les interruptions sont autorisées après l'exécution de l'instruction SUIVANT EI.

Indicateurs : SZ H PNC
aucun effet

Remarque : Si vous interdisez les interruptions dans une routine, n'oubliez pas de les autoriser à nouveau à la fin, sinon vous n'aurez plus le contrôle de l'ordinateur, car il n'y aura pas de scrutation clavier.

NOP

(*Pas d'effet)*

Effet : Aucun.

Indicateurs : SZ H PNC
aucun effet

Remarque : Cette instruction (très paresseuse !) ne sert qu'à produire de petits délais (NOP dure 1 cycle machine, soit environ 1 microseconde, ce n'est pas bien long), ou à réserver des octets dans un programme pour une future modification.

HALT

(Gel de l'unité centrale)

Description : Le Z80 arrête son fonctionnement et exécute des NOP jusqu'à recevoir une interruption ou une réinitialisation.

Indicateurs : SZ H PNC
aucun effet

Remarque : Cette instruction peut servir à générer des délais assez importants. La fréquence des interruptions sur MSX européen est de 50HZ ; donc, si vous faites une boucle qui exécute 50 fois HALT, vous créez un délai d'environ 1 seconde.

7

Initiation à la programmation

Le chapitre précédent vous a fourni l'ensemble des instructions existant au sein du Z80. Nous allons à présent apprendre à nous servir de ces instructions, à les ordonner, de manière à construire de petits programmes permettant la résolution de tâches simples. Les exemples que nous avons pris dans ce chapitre ont été classés par difficulté croissante, nous vous conseillons donc de les aborder dans l'ordre.
Avant de concevoir un programme, quel qu'il soit, il faut construire un algorithme (cf. chapitre Principes de bases, partie Algorithme, organigramme) permettant la résolution du problème envisagé. Cette phase préalable est indispensable ; sans elle, votre programme a toutes les chances de ne pas « tourner ». C'est pourquoi, pour chacun des problèmes que nous vous soumettrons dans ce chapitre, nous commencerons par une courte analyse, puis nous écrirons un ou plusieurs algorithmes, et enfin, nous écrirons le ou les programmes correspondants.

- *Chercher une valeur donnée dans une table*

L'algorithme et l'organigramme de ce problème ont déjà été donnés dans le chapitre intitulé Principes de base. Nous nous contenterons de construire le programme correspondant.

Supposons que notre table commence à l'adresse 1000H et se termine à l'adresse 1400H et que la valeur que l'on cherche au sein de cette table soit 42H. Il y a ainsi 400H octets dans notre table.

Choisissons le registre HL pour « montrer du doigt » l'élément à comparer.

Choisissons BC comme compteur de boucle.

En retour, on veut dans le double registre HL l'adresse de la première occurrence, si l'on a trouvé la donnée dans la table, et 0 si la donnée n'a pas été trouvée dans la table.

Le programme est alors :

```
        LD HL,1000H       ; HL : = adresse début table
        LD BC,400H        ; BC : = nombre d'élts de la table
BOU     LD A,(HL)         ; A = (HL)
        CP 42H            ; A = 42H ?
        JP Z,FIN          ; On a trouvé
        INC HL            ; HL : = HL + 1
        DEC BC            ; BC : = BC - 1
        LD A,B            ; A : = B
        OR C              ; A V. C
        JP NZ,BOU         ; BC ≠ 0, on continue
        LD HL,0           ; Elt non trouvé
FIN     RET               ; Retour
```

Bien évidemment, ce programme n'est pas optimal. Cependant, si vous l'avez bien compris, vous avez déjà fait un pas en avant. A noter que ce programme comporte une petite astuce permettant de tester si un double registre est nul : l'instruction LD A,B transfère le contenu du registre B dans l'accumulateur. L'instruction suivante OR C compare le contenu du registre C à celui de l'accumulateur (qui contient en fait B). L'indicateur Z est donc positionné si et seulement si B=C=0 (cf. tableau de vérité du OU chapitre Arithmétique et logique). Cette astuce permet de tester si un registre double est nul ou non en deux instructions.

A titre d'exemple, nous vous donnons maintenant le même programme mais optimisé :

```
        LD HL,1000H       ; HL : = adresse début table
        LD BC,400H        ; BC : = nombre d'élts de la table
        LD A,42H          ; A : = donnée à trouver
        CPIR              ; Comparaison avec incrémentation
        CP (HL)           ; A = (HL) ?
        JP Z,FIN          ; Si oui FIN
        LD HL,0           ; Sinon HL 8 = 0
FIN     RET               ; Retour
```

Ici, l'emploi de l'instruction CPIR facilite grandement les choses : on n'a plus à se soucier des pointeurs, ni à les incrémenter. Seules l'adresse de début de la table, sa longueur et la donnée à trouver doivent être respectivement mises dans HL, BC et A. L'instruction de comparaison qui suit le CPIR sert à mettre HL à 0 dans le cas ou la donnée n'aurait pas été trouvée.

- *Boucle d'attente*

L'assembleur est rapide, parfois même trop rapide. Lorsque l'on désire transmettre un résultat à l'utilisateur (sur l'écran par exemple), il est nécessaire de laisser ce résultat imprimé un certain temps. De même, si vous réalisez un jeu en ASSEMBLEUR, vous vous apercevrez bien vite que si vous affichez les motifs sans vous préoccuper de les laisser à l'écran un certain temps, vous ne voyez absolument rien ! Il est donc nécessaire d'avoir recours à des boucles d'attente (aussi appelées **boucles de temporisation**) qui seront là pour « perdre » du temps et par la même permettre à l'utilisateur de voir ce qui se passe.

Le programme qui suit est simple et court. Il a été écrit pour perdre un temps fonction d'un nombre se trouvant initialement dans l'accumulateur.

```
TEMPO    NOP              ; Ne rien faire
         DEC A            ; A : = A - 1
         JR NZ,TEMPO      ; Recommencer si A ≠ 0
         RET              ; Retour
```

Le calcul du temps perdu par cette boucle est simple, et vous pouvez d'ailleurs vous exercer à le refaire avec l'aide de l'Annexe 5. Une boucle prend 20 cycles (4 + 4 + 12) ; la dernière boucle en prend 25 (4 + 4 + 7 + 10). Si vous avez par exemple 201 dans l'accumulateur, cette boucle prendra 200*20 + 25 soit 4025 cycles en tout. Avec notre horloge à 4 Mhz, ceci fait 1005 microsecondes, soit environ un millième de seconde. Vous pouvez bien sûr, si vous voulez produire des délais d'attente plus importants, soit inclure cette boucle dans une autre, soit vous servir d'un double registre à la place de l'accumulateur. Dans ce dernier cas, prenez bien garde à comparer EXPLICITEMENT le contenu de votre double registre avec 0 car la décrémentation d'un double registre n'affecte pas les indicateurs d'état.

Pour produire un délai d'attente, on peut utiliser l'instruction HALT (cf. Jeu d'instructions) qui suspend l'exécution jusqu'à la prochaine interruption envoyée par le VDP.

- *Multiplication 8 bits par 8 bits*

Pour cet exemple, nous fournirons deux algorithmes totalement différents. Le premier est simple à comprendre mais ne satisfait pas à certains objectifs de la programmation. Nous le présentons donc uniquement dans un esprit pédagogique pour montrer comment on évalue un programme. Le second, beaucoup plus complexe, est très satisfaisant.

Premier algorithme :

Vous avez à réaliser la multiplication de deux nombres de 8 bits. Le résultat tiendra donc sur 16 bits. Soit X et Y les deux nombres à multiplier. Un algorithme simple pour multiplier vos deux nombres est d'ajouter Y fois le nombre X

Voici l'algorithme :

```
1. Mettre le résultat à zéro
2. Si X = 0 alors aller en 3
          sinon faire résultat : = résultat + Y
                      X : = X-1
                      retourner en 2
3. Fin de la multiplication
```

Le programme est alors très simple à écrire. Choisissons pour la donnée X l'accumulateur, pour la donnée Y le registre C, et HL pour le résultat.

```
MULT    LD A,donnée1
        LD B,0
        LD C,donnée2
        LD HL,0
BOU     OR A              ; A = 0 ?
        JR Z,FIN          ; Si oui finir
        ADD HL,BC         ; HL : = HL + BC
        DEC A             ; A : = A-1
        JP BOU            ; Boucler
FIN     RET               ; Retour
```

A noter que le OR A permet de savoir si A est nul en une instruction. En effet OR A produit un OU logique en l'accumulateur et lui même. Le résultat de ce OU est donc 0 si et seulement si A=0.

Pourquoi ce programme est-il mauvais ?

La raison en est simple : lorsqu'on écrit un programme, on aime bien connaître le temps qu'il va mettre pour s'exécuter. Ici, le temps qu'il met pour effectuer la multiplication est fonction des opérandes : en effet, si l'opérande X vaut 0 il n'y aura aucune boucle qui sera exécutée et si A vaut 255, il y aura 255 boucles effectuées. L'opérande X influence donc le temps d'exécution de manière considérable puisque selon sa valeur, le temps varie entre 57 cycles dans le meilleur des cas (A = 0) et 9237 dans le pire des cas (A = 255). Ceci n'étant pas acceptable, cet algorithme est donc à rejeter.

En voici un autre :

Observons, pour l'illustrer, le principe de la multiplication que l'on apprend dans les petites classes : il s'agit de multiplier les nombres 134 et 102 :

```
X :    134
Y :    102
      ----
       268
      000
     134
      ----
     13668
```

Faisons-le :

```
X :        10000110
Y :        01100110
         ----------
           00000000
          10000110
         10000110
        00000000
       00000000
      10000110
     10000110
    00000000
   ----------------
    011010101100100
```

Nous pouvons remarquer que le nombre de la deuxième ligne est en fait 2*X (puisqu'il s'agit de X décalé d'une position vers la gauche), que le nombre de la troisième ligne est 4*X etc. D'autre part, on remarque que les lignes où il y a 0 correspondent à des bits nuls de l'opérande Y. Notre algorithme va donc consister à multiplier l'un des opérandes successivement par 1, 2, 4, 8, etc. pendant que l'on récupère 1 à 1 tous les bits de l'autre opérande. Si le bit en question est 1, on ajoutera alors le premier opérande (alors multiplié par une puissance de deux) au résultat, sinon on ne fera rien. Cette tâche devra être effectuée pour les huits bits du deuxième operande. L'algorithme s'écrit alors :

1. Mettre le résultat à zéro
   Mettre un compteur à 8
2. Décaler le second opérande à droite et récupérer le bit faible
3. Si ce bit vaut 0 alors aller en 4, sinon faire résultat : = résultat + opérande1
4. Multiplier l'opérande 1 par 2
5. Décrémenter le compteur
6. Si compteur ≠ 0 aller en 2

Choisissons HL pour implanter le résultat, A pour le second opérande, BC pour l'opérande 1 et E pour le compteur. Le programme devient alors :

```
MULT    LD HL,0        ; HL : = 0
        LD E,8         ; E : = 8
        LD A,op2       ; A : = op2
        LD B,0         ; B : = 0
        LD C,op1       ; C : = op1
BOU     SRL A          ; Décalage à droite de op2
        JP NC,SUITE    ; Si bit ≠ 1, continuer
        ADD HL,BC      ; Sinon résultat : = résultat + op1
SUITE   SLA C          ; Décalage à gauche
        RL B           ; du double registre BC
        DEC E          ; E : = E-1
        JP NZ,BOU      ; Boucler si E ≠ 0
        RET            ; Retour
```

Ce programme est beaucoup mieux que le précédent car son temps d'exécution est en moyenne plus court et, de plus, il ne dépend presque pas des opérandes. En effet, il prend entre 432 (cas où tous les bits de op2 sont à zéro) et 520 cycles (cas où tous les bit de op2 sont à 1). Cependant, il est améliorable. Nous vous en offrons à présent une version plus rapide (mais toujours basée sur le principe de décalage-addition) :

```
MULT    LD H,op1        ; H : = op1
        LD L,0          ; L : = 0
        LD E,op2        ; E : = op2
        LD D,0          ; D : = 0
        LD B,8          ; B : = 8
BOU     ADD HL,HL       ; Décalage à gauche de op1
        JR NC,SUITE     ; Si bit = 0 continuer
        ADD HL,DE       ; Sinon résultat : = résultat + op2
SUITE   DJNZ BOU        ; Si B ≠ 0 boucler
        RET             ; Retour
```

Ce dernier programme prend entre 327 et 375 cycles, ce qui représente encore un gain de vitesse assez appréciable par rapport au précédent. Son fonctionnement est assez particulier puisque, à un instant donné, les bits restants de op1 et le début du résultat se trouvent tous les deux dans HL. Pour mieux comprendre ce programme, nous vous conseillons de prendre deux nombres au hasard (compris bien sûr entre 0 et 255) et de faire tourner le programme à la main en représentant à chaque étape le contenu du registre HL.

■ *Division 16 bits par 8 bits*

L'algorithme de la division est très proche de celui de la multiplication, à ceci près, que l'on effectuera des soustractions à la place des additions. Cette technique peut être qualifiée de **décalage-soustraction.** Comme dans le dernier programme de multiplication, quotient et partie restante d'un des opérandes seront dans le même registre. Voici l'algorithme :

1. Résultat : = 0
   Compteur : = 8
   diviseur : = diviseur * 256
2. Soustraire diviseur de dividende
   quotient : = quotient + 1
3. Si reste positif, alors aller en 4,
   sinon faire ajouter diviseur au dividende
   quotient : = quotient − 1
4. Décaler le dividende à gauche
5. Compteur : = compteur − 1
6. Si compteur ≠ 0, aller en 2

Choisissons le registre HL pour mettre le dividende et le résultat, B comme compteur et DE pour mettre 256 × diviseur.

Le programme devient :

```
DIV     LD B,8          ; B : = 8
        LD C,0          ; C : = 0
        LD D,divis      ; D : = diviseur
        LD HL,divid     ; HL : = dividende
BOU     AND A           ; Carry : = 0
        SBC HL,DE       ; Dividende : = dividende – diviseur
        INC HL          ; Quotient : = quotient + 1
        JP P,SUITE      ; Si reste positif, faire la suite
        ADD HL,DE       ; Restauration du dividende
        DEC HL          ; et du quotient
SUITE   ADD HL,HL       ; Décalage à gauche du dividende
        DJNZ BOU        ; Boucler si B ≠ 0
        RET             ; Retour
```

A noter que l'instruction AND A est utilisée pour mettre la Carry à zero car il n'y a pas de soustraction (sur 16 bits) sans retenue sur le Z80, mais seulement une soustraction avec retenue. La retenue doit donc impérativement être mise à 0 pour que le résultat ne soit pas faussé.

■ *Conversion ASCII ←→ DCB*

Les conversions constituent un élément indispensable en programmation. En effet, il arrive très souvent que l'on saisisse des nombres sur un clavier, que l'on récupère leur code ASCII et que l'on veuille alors faire des calculs sur ces nombres. Le calcul en ASCII étant impossible, il faut, soit les convertir en binaire, soit en DCB. De même, il peut vous arriver d'avoir fait des calculs et de vouloir les imprimer à l'écran ou sur une imprimante. Vous avez alors besoin de routines de conversion DCB-ASCII ou HEXA-ASCII. Nous avons choisi ici de vous proposer des routines de conversion ASCII-DCB.

DCB → ASCII

Supposons que l'on ait dans l'accumulateur un nombre code en DCB et que l'on veuille avoir les deux codes ASCII (correspondant à chacun des deux quartets) dans deux autres registres, par exemple B et C.

L'algorithme de conversion DCB → ASCII est très simple : en DCB, un chiffre est codé sur un quartet par la valeur qu'il représente (1 est codé 0001 soit 1, 2 est codé 0010 soit 2, etc.). Remarquons qu'en ASCII, 0 est codé 30H, 1 est codé 31H ... 9 est codé 39H. Pour convertir un chiffre DCB en le code ASCII lui correspondant, il suffit donc de lui ajouter 30H. L'algorithme est donc le suivant :

1. Conserver le quartet fort du nombre
2. Ajouter 30H à ce quartet et sauvegarder le résultat
3. Conserver le quartet faible du nombre
4. Ajouter 30H à ce quartet et sauvegarder le résultat

Le programme est alors :

```
DCB-A   LD C,A      ; Conserver le nombre initial
        SRL A       ; Conservation
        SRL A       ;     du
        SRL A       ;   quartet
        SRL A       ;     fort
        ADD 30H     ; Addition
        LD B,A      ; Stockage du premier code ASCII
        LD A,C      ; Restauration du nombre
        AND 0FH     ; Conservation du quartet faible
        ADD 30H     ; Addition
        LD C,A      ; Stockage du second code ASCII
        RET         ; Retour
```

□ *Conversion ASCII-DCB*

Si maintenant on veut faire le contraire, c'est très simple. Voici l'algorithme :

1. Soustraire 30H du premier code ASCII et conserver ce résultat en temps que quartet fort.
2. Soustraire 30H du deuxième code ASCII et conserver ce résultat en temps que quartet faible.
3. Rassembler les deux quartets obtenus pour en faire un octet.

Le programme devient (avec les mêmes conventions que précédemment) :

```
A-DCB   LD A,B      ; Code ASCII du quartet fort
        SUB 30H     ; Soustraction
        SLA A       ; Décalage
        SLA A       ; du nombre
        SLA A       ; dans le
        SLA A       ; quartet fort
        LD B,A      ; Sauvegarde du quartet fort
        LD C,A      ; Code ASCII du quartet faible
        SUB 30H     ; Soustraction
        ADD B       ; Rassemblement des deux quartets
        RET         ; Retour
```

APPLICATION AUX MSX | 2

8

Organisation générale des MSX

LES DIVERS PROCESSEURS DES MSX

Les MSX sont architecturés autour du processeur Z80 de chez Zilog et dont le jeu d'instructions vient d'être vu en détail. Ce processeur est assez puissant, mais il ne serait pas intéressant (et même parfois impossible) de lui laisser tout faire. C'est pourquoi les constructeurs ont décidé de le décharger de certaines tâches très spécifiques en implantant d'autres processeurs. On peut donc trouver sur le circuit imprimé d'un MSX trois autres processeurs qui sont :

— Un TMS 9929
— Un AY-3-8910
— Un PPI 8255

Tous ces noms ne vous disent certainement rien (si cela n'est pas le cas, vous pouvez passer au chapitre suivant). Ce chapitre a donc pour but de vous expliquer à quoi servent ces différents organes.

Le Z80 doit inévitablement communiquer avec ces divers processeurs. Il le fait par le principe d'entrée sortie que nous allons voir dès à présent :

Définissons d'abord les termes : on appelle **entrée** toute action qui consiste à amener des données dans le système depuis des organes extérieurs (clavier, disque, etc.). De même, on appelle **sortie** toute action qui consiste à transférer des données vers des organes extérieurs (imprimante, disque, etc.) depuis le système (microprocesseur et mémoire). Ici nous ne parlerons que des entrées/sorties effectuées entre le Z80 et les trois processeurs

cités ci-dessus. Ces entrées sorties se font par l'intermédiaire de ports d'entrées/sorties sur lesquels le Z80 peut envoyer des données (cas d'une sortie) ou en recupérer (cas d'une entrée). Chaque processeur possède son propre réseau de registres d'entrées/sorties sur lequel il reçoit ou transmet des données. Le Z80, quant à lui, possède des instructions lui permettant de lire ou d'écrire sur un port donné. Ces instructions sont IN et OUT ainsi que toutes leurs variantes. Nous vous donnerons plus loin un tableau rassemblant les adresses où s'effectuent des entrées/sorties ainsi que les organes auxquels elles sont réservées. En attendant, regardons un peu ce que ces autres processeurs sont capables de faire.

- *Le TMS 9929*

Ce nom étant trop barbare, nous ne nommerons VDP (pour *Video Display Processor*) car c'est en effet lui qui se charge de la gestion de l'écran. Ce processeur dispose de sa propre mémoire (16 Ko) qu'il gère (du mieux qu'il peut !!!) mais qui n'est en aucun cas directement accessible au Z80. Ce processeur possède 9 registres internes que l'on peut lire ou charger par des procédures d'entrées sorties. Sa mémoire lui permet d'afficher des dessins dans des modes allant du texte à la haute résolution (256 × 192) et ceci avec des couleurs diverses. Ses registres internes lui permettent de gérer les divers modes graphiques (4 au total) ainsi que les **sprites**. Nous étudierons le fonctionnement de ce processeur en détail dans le chapitre 9.

- *Le AY-3-8910*

Ce nom étant lui aussi trop barbare, nous le nommerons quant à lui PSG (pour *Programmable Sound Generator*) car c'est lui qui se charge de la partie son. Il possède 16 registres internes qui lui permettent de gérer 3 voies indépendantes. La communication avec ce processeur se fait elle aussi par des registres d'entrées/sorties. Il faut noter que l'utilisation de ce processeur permet de décharger le Z80 mais aussi de faire 2 tâches simultanément : en effet, dès que les paramètres nécessaires pour jouer une ou plusieurs notes lui ont été transmis, le PSG continue à jouer (tout seul comme un grand) jusqu'à ce qu'on lui en transmette d'autres. On peut donc, pendant qu'il joue une ou plusieurs notes, continuer une autre tâche sans s'occuper de lui. Ceci n'est pas réalisable sur les systèmes qui ne possèdent pas de processeur sonore et où le fait de jouer de la musique empêche de faire autre chose en même temps.

- *LE PPI 8255*

Nous le nommerons tout simplement PPI (pour *Programmable Port Interface*). Ce circuit est spécialisé pour la gestion de la cassette et du clavier. Il sert aussi à la commutation des mémoires que nous étudierons dans le chapitre 10.

Comme nous l'avons dit, chacun de ces processeurs communique avec le Z80 par des entrées/sorties qui se font sur des ports dont nous donnons ici un tableau. (Un tableau complet de tous les ports d'entrées/sorties sera donné en annexe 4.)

Proc	ad	Fonction
VDP	98H	Ecriture/Lecture des registres du VDP
VDP	99H	Ecriture registres VDP
PSG	A0H	Ecriture du numéro du registre du PSG désiré
PSG	A1H	Ecriture registres PSG
PSG	A2H	Lecture ports E/S PSG
PPI	A8H	Ecriture/Lecture port A du PPI
PPI	A9H	Ecriture/Lecture port B du PPI
PPI	AAH	Ecriture/Lecture port C du PPI
PPI	ABH	Ecriture/Lecture registre commande du PPI

Les trois chapitres qui suivent vont étudier chacun en détail un de ces trois processeurs. Nous vous conseillons d'y prêter une attention particulière (surtout en ce qui concerne le VDP) car c'est grâce à ces processeurs annexes que vous pourrez extraire la quintessence de votre MSX.

L'ORGANISATION MÉMOIRE DES MSX

Les MSX possèdent une certaine quantité de mémoire, variable selon les modèles et les constructeurs.

La mémoire écran est de 16 ko quel que soit le modèle que vous avez. Nous avons vu (partie sur le VDP) que ces 16 ko n'étaient pas accessibles directement car ils étaient gérés par le VDP. D'autre part, ces 16 ko ne sont pas comptés dans la totalité de la mémoire (lorsque vous avez un MSX 32 ko, il y a en fait 16 Ko supplémentaires qui correspondent à la mémoire écran également appelée VIDEORAM ou encore VRAM).

La mémoire morte (ROM pour *Read Only Memory*) est de 32 Ko quel que soit le modèle que vous possédez. Cette ROM contient le BASIC MICROSOFT et encore d'autres choses. Nous l'étudierons en détail au cours du chapitre 12.

La mémoire vive (RAM pour *Random Acces Memory*) est variable selon les modèles et les constructeurs. Les RAM que l'on trouve sur MSX sont de 8, 16, 32 ou 64 Ko selon les modèles. C'est cette partie qui est allouée aux programmes et dont l'utilisateur peut se servir à sa guise. Voyons un peu comment cette mémoire est organisée.

La mémoire directement adressable par le Z80 est de 64 Ko. En effet, ce processeur possède un PC et un bus d'adresses sur 16 bits (cf. chapitre Organisation interne du Z80) et ne peut donc gérer qu'un maximum de 65536 octets, soit exactement 64 Ko (cf. définition du Ko). Le MSX divise cette mémoire en 4 parties égales de 16 Ko que l'on nomme BANKS. Chacun de ces BANKS est capable de choisir un SLOT parmi quatre. Nous verrons dans le chapitre sur le PPI comment se fait cette sélection de SLOTS pour chacun des quatre BANKS. La configuration est toujours la suivante :

Le BANK 0 occupe les adresses de 0000H à 3FFFH
Le BANK 1 occupe les adresses de 4000H à 7FFFH

Le BANK 2 occupe les adresses de 8000H à BFFFH
Le BANK 3 occupe les adresses de C000H à FFFFH

Les slots sont numérotés de 0 à 3 (comme les BANKS). Ce système de SLOTS permet d'augmenter la mémoire de manière considérable (les constructeurs annoncent que celle-ci est extensible jusqu'à 1 Mo, soit 1024 Ko !!!). Cependant, cette mémoire n'est jamais accessible en entier à un instant donné. Si l'on veut accéder à toute cette mémoire dans un MSX « gonflé », il faut faire des manipulations supplémentaires des BANKS et des SLOTS, ce que nous apprendrons dans le chapitre consacré au PPI. Voici un petit tableau qui résume l'organisation matérielle de la mémoire de votre MSX.

Organisation mémoire

	Slot 0	Slot 1	Slot 2	Slot 3
0000H – Bank 0 – 4000H	ROM	CARTOUCHES		RAM
4000H – Bank 1 – 8000H	ROM			RAM
8000H – Bank 2 – C000H	RAM		DISK SOFTWARE	
C000H – Bank 3 – FFFFH	RAM			

L'étude complète des diverses parties de cette mémoire (ROM, région de communication) sera faite au chapitre 12. Cependant, nous pouvons dès à présent vous donner une idée de la manière dont la ROM et la région de communication sont organisées.

La ROM comporte essentiellement deux parties : une partie que nous nommerons BIOS qui se trouve logée de l'adresse 0000H à l'adresse 015BH, et une partie où se trouve le BASIC qui est logée de l'adresse 01B6H à l'adresse 7FFFH. Le BIOS contient un ensemble de routines qui sont COMMUNES à tous les MSX. C'est donc les routines du BIOS que vous utiliserez plutôt que celles du BASIC dans le but de réaliser un programme compatible à tous les MSX actuels et futurs. En effet, la partie BASIC peut être sujette à des modifications futures (pour l'instant, tous les BASICS MSX sont identiques). Le BIOS contient un ensemble de **crochets** dont nous vous donnerons le détail au chapitre 12. C'est par ces crochets que vous devrez passer car ce sont eux qui garantissent le standard MSX.

La région de communication (RC) se trouve en haut de la mémoire et contient deux parties. La première rassemble un ensemble de variables système grâce auxquelles on peut obtenir des informations sur l'état de la machine à un moment donné ou modifier celui-ci. On y trouve les variables contenant la couleur de l'encre ou du fond, les valeurs des adresses table du VDP, la borne supérieure de la RAM, etc. La deuxième partie contient un ensemble de vecteurs crochet dont beaucoup ne servent que lorsqu'on utilise le disque.

Tout ceci sera revu en détail dans le chapitre 12.

9

Le VDP

Le périphérique le plus important est sans doute l'écran de visualisation. Dans le système MSX, l'écran est géré par un processeur spécialisé, le *Video Display Processor* (VDP) TMS 9929 de Texas Instrument. Le VDP dispose de sa propre mémoire, qu'il gère entièrement seul, sans l'aide du Z80 et que ce dernier ne peut adresser directement. C'est là la première caractéristique du VDP : sa mémoire, appellée VIDEORAM ou VRAM ne prend aucune place sur le bus d'adresses du Z80, laissant ainsi plus d'espace pour les programmes. C'est un avantage par rapport aux machines dont la mémoire écran fait partie de la RAM centrale. Le VDP dégage aussi le Z80 de toutes les servitudes vidéo.

En contrepartie, le VDP est vu par le Z80 comme un périphérique uniquement accessible par les instructions IN et OUT, et le jeu d'instructions du Z80 est inopérant dans la Vidéoram.

Le fonctionnement du VDP et la structure de la Vidéoram sont assez compliqués ; nous allons les étudier progressivement.

L'image créée par le VDP est composée de 34 plans successifs (les plans de chacun des 32 SPRITES, celui des textes et graphiques, celui du fond). Ces plans sont numérotés (le plan numéro 0 est le plus « près » de l'observateur). Dans le cas où plusieurs plans contiennent des objets au même emplacement, c'est l'objet du plan de numéro le plus bas qui est visualisé. C'est ce qu'on appelle l'**ordre de priorité.**

Les 32 premiers plans peuvent contenir un (et un seul) sprite. Un sprite ou **lutin** est un dessin 8×8, 16×16 ou 32×32 points (les points sont soit allumés dans une couleur définie, soit transparents). Le reste du plan est transparent. L'emplacement d'un sprite dans son plan est déterminé par ses coordonnées, ce qui rend son déplacement aisé. Nous détaillerons tout ceci dans le paragraphe sur les sprites.

Les 2 derniers plans peuvent être gérés de différentes façons, suivant le **mode** dans lequel on se trouve.

Il existe 4 modes de fonctionnement dans le VDP :

— Mode TEXTE : il correspond à SCREEN 0
— Mode GRAPHIQUE 1 : il correspond à SCREEN 1
— Mode GRAPHIQUE 2 : il correspond à SCREEN 2
— Mode MULTICOLORE : il correspond à SCREEN 3

LES TABLES DU VDP

La Vidéoram occupe 16 Ko. Elle se divise en un certain nombre de tables dont les fonctions et les adresses dépendent du mode de fonctionnement.

Pour afficher quelque chose à l'écran (en dehors des sprites), il faut :

□ dire ce que l'on veut afficher : l'écran affiche des CONFIGURATIONS de 6×8 points (mode texte) ou 8×8 (autres modes). Les configurations sont définies dans une table, la Table Génératrice des Configurations (TGC). Chaque configuration occupe 8 octets dans cette table : la première configuration va de l'octet 0 à l'octet 7, la seconde, de l'octet 8 à l'octet 15, etc. On peut donc facilement accéder à une configuration particulière par son numéro d'ordre dans la table, que l'on appelle son NOM. Le nombre des configurations possibles varie avec les modes ; en mode texte ou graphique 1, ces configurations ne sont autres que le jeu de caractères (les 256 caractères alphanumériques et semi-graphiques que vous connaissez) dont les noms sont leurs codes ASCII habituels.

□ dire où l'on veut l'afficher : l'écran est divisé en 40×24 cellules (mode texte), 32×24 cellules (autres modes). Dans chaque mode, une autre table, appelée Table de Nom des Configurations (TNC), longue d'autant d'octets qu'il y a de cellules, contiendra les noms (en fait les numéros) des configurations qu'il faut afficher. On met dans le premier octet de la TNC le nom de la configuration à afficher dans la première cellule, dans le deuxième octet, le nom de la configuration à afficher dans la deuxième cellule, et ainsi de suite.

□ dire dans quelle couleur on veut l'afficher : suivant les modes, on se sert ou non d'une troisième table, la Table des Couleurs (TC) qui définit de quelle(s) couleur(s) seront affichées les différentes configurations.

Les sprites, eux, sont définis grâce à deux autres tables :

□ La Table Génératrice des Sprites (TGS) contient les dessins des sprites, comme expliqué plus loin.

□ La Table des Attributs des Sprites (TAS) définit la position, le numéro, et la couleur de chaque sprite.

Il y a donc 5 tables par mode au total :

— la table des noms des configurations (TNC)
— la table génératrice des configurations (TGC)
— la table des couleurs (TC)
— la table génératrice des sprites (TGS)
— la table des attributs des sprites (TAS)

Les localisations au sein de la Vidéoram des différentes tables varient avec les modes et sont déterminées par le contenu des registres internes du VDP.

LES REGISTRES INTERNES DU VDP

Le VDP a 9 registres internes ; dans 8 d'entre eux (R0 à R7), on ne peut qu'écrire, le 9e (registre d'état) ne peut être que lu.

Leurs contenus déterminent le mode et les adresses des différentes tables. Ils sont chargés par la ROM à l'initialisation de chaque mode avec les adresses standard des tables correspondant à ce mode. Il est toutefois possible de les modifier, par exemple pour obtenir plusieurs pages écran, ou plusieurs jeux de caractères.

Examinons-les :

- *Le registre 0 (R0)*

Seul son bit 1 (les bits sont toujours numérotés de 0 à 7 à partir de la droite) est utilisé ; il commande le passage en mode graphique 2 : il est à 1 dans ce mode, à 0 sinon. On l'appelle le bit M3. Les autres bits doivent être à 0.

- *Le registre 1 (R1)*

Il contrôle différents paramètres : le bit 0 est le facteur d'agrandissement des sprites : 0 indique la taille normale, 1 indique l'agrandissement (×2, donc sprites sur 16×16 ou 32×32 points).

Le bit 1 indique la taille des sprites : 0 = taille 8×8, 1 = taille 16×16 points.

Le bit 2 est toujours à 0.

Le bit 3 commande le passage en mode multicolore : il est à 1 dans ce mode, à 0 dans les autres cas. On l'appelle le bit M2.

Le bit 4 commande le passage en mode texte : il est à 1 dans ce mode, à 0 dans les autres cas. On l'appelle le bit M1.

Le bit 5 autorise (1) ou interdit (0) les interruptions.

Le bit 6 autorise (1) ou interdit (0) l'affichage de l'image.

Le bit 7 est toujours à 1.

■ *Le registre 2 (R2)*

Les bits b0 à b3 sont les 4 bits les plus significatifs de l'adresse de la Table de Nom des Configurations (quelque soit le mode) ; les autres bits doivent être à 0. La Vidéoram faisant 16 Ko, une adresse se code sur 14 bits (0 à 3FFFH). On pourra donc, à l'aide du registre 2 disposer la TNC n'importe où dans la Vidéoram par pas de 1K (1024). C'est logique car le bit 0 de R2 est le bit 10 de l'adresse complète, il a donc le poids 1024 dans cette adresse. L'adresse effective de la TNC vaut le contenu de R2 multiplié par 1024, et il y a 16 possibilités pour la placer.

Essayez de bien comprendre ce mécanisme, car tous les registres et toutes les tables fonctionnent de la même façon, bien qu'avec un nombre de bits différent.

■ *Le registre 3 (R3)*

Il contient les 8 bits les plus significatifs de l'adresse de la Table de Couleurs. On peut donc mettre la TC n'importe où dans la Vidéoram par pas de 64 octets. L'adresse complète de la TC vaut donc R3×64, et il y a 256 endroits possibles.

En mode graphique 2, la TC ayant 6144 octets de long (voir plus loin), seul le bit 7 de R3 est utile, les autres sont à 1 ; s'il est à 0 (R3=127), la TC commence à l'adresse 0, s'il est à 1 (R3=255), la TC commence à l'adresse 2000H (8192).

■ *Le registre 4 (R4)*

Ses bits b0 à b2 sont les 3 bits les plus significatifs de l'adresse de la Table Génératrice des Configurations ; les autres doivent être à 0. On peut donc mettre la TGC n'importe où dans la Videoram par pas de 2K (2048), et il y a 8 endroits possibles. L'adresse complète de la TGC vaut R4×2048.

En mode graphique 2, la TGC ayant 6144 octets de long (voir plus loin), seul le bit 2 est actif, les deux premiers sont à 1 ; s'il est à 0 (R4=3), la TGC commence en 0, s'il est à 1 (R4=7), elle commence en 2000H (8192).

■ *Le registre 5 (R5)*

Ses bits b0 à b6 sont les 7 bits les plus significatifs de l'adresse de la Table des Attributs des Sprites ; le bit 7 doit être à 0. On peut donc mettre la TAS n'importe où dans la Vidéoram par pas de 128 octets, et il y a 128 endroits possibles. L'adresse complète de la TAS vaut R5×128.

■ *Le registre 6 (R6)*

Ses bits b0 à b2 sont les 3 bits les plus significatifs de l'adresse de la Table Génératrice des Sprites ; les autres doivent être à 0. On peut donc mettre la TGS n'importe où dans la Vidéoram par pas de 2K (2048). L'adresse complète de la TGS vaut R6×2048, et il y a 8 endroits possibles.

■ *Le registre 7 (R7)*

Il contient l'information couleur pour le mode texte. Vous savez que 16 couleurs existent ; on peut donc les coder sur 4 bits. Voici le codage utilisé :

Binaire	Décimal	Hexa	Couleur
0000	0	0	Transparent
0001	1	1	Noir
0010	2	2	Vert moyen
0011	3	3	Vert clair
0100	4	4	Bleu foncé
0101	5	5	Bleu clair
0110	6	6	Rouge foncé
0111	7	7	Cyan
1000	8	8	Rouge moyen
1001	9	9	Rouge clair
1010	10	A	Jaune foncé
1011	11	B	Jaune clair
1100	12	C	Vert foncé
1101	13	D	Magenta
1110	14	E	Gris
1111	15	F	Blanc

Ce codage sera d'ailleurs le même dans la table des couleurs, pour les modes qui l'utilisent.

Les bits b0 à b3 de R7 définissent en mode texte la couleur du fond, c'est-à-dire la couleur des 0 de la TGC ; dans les autres modes, ils définissent la couleur du bord de l'écran. Les bits b4 à b7 définissent en mode texte la couleur de l'encre, soit la couleur des 1 de la TGC.

■ *Le registre d'état*

Ce registre est à lecture seule ; il donne des indications sur les interruptions et les sprites.

Les bits b0 à b4 contiennent le numéro du sprite qui a provoqué la mise à 1 du bit b6 (présence de 5 sprites ou plus sur une même horizontale).

Le bit b5 est le Flag de coïncidence, il est mis à 1 si 2 ou plusieurs sprites ont au moins un point commun (mais on n'a pas les numéros de ces sprites).

Le bit b6 est mis à 1 si 5 sprites ou plus sont présents sur une même horizontale. Il est remis à 0 par une lecture du présent registre d'état.

Le bit b7 est le Flag d'interruption. Il est mis à 1 à la fin du balayage de l'écran, et est remis à 0 par une lecture du présent registre d'état.

Récapitulation sur les bits M1, M2 et M3 :

M1	M2	M3	Mode obtenu
0	0	0	Graphique 1
0	0	1	Graphique 2
0	1	0	Multicolore
1	0	0	Texte

Toutes les autres combinaisons donnent des mélanges de modes difficilement exploitables, mais vous pouvez toujours essayer...

LA RÉGION DE COMMUNICATION (RC)

Dans de nombreuses routines internes, la ROM a besoin de savoir à quelles adresses de la VRAM se trouvent les tables dans les différents modes, le mode courant (actuel) de l'écran et les adresses courantes des tables dans le mode actuel.

Puisque les registres R0 à R7 du VDP sont à écriture seule, la ROM ne peut les lire pour obtenir ces informations. La seule solution est de stocker ces adresses dans la région de communication au moment de leur écriture dans les registres. Les adresses F3B3H à F3D9H et F91FH à F928H de la RAM servent à cela ; les adresses F3DFH à F3E6H conservent les contenus des 8 registres du VDP. D'autres adresses de la région de communication seront données plus loin (cf. chapitre 12).

F3B3-F3B4	adresse	de la	TNC	Texte
F3B5-F3B6	"	"	TC	Texte (inutilisée)
F3B7-F3B8	"	"	TGC	Texte
F3B9-F3BA	"	"	TAS	Texte (inutilisée)
F3BB-F3BC	"	"	TGS	Texte (inutilisée)
F3BD-F3BE	"	"	TNC	Grap.1
F3BF-F3C0	"	"	TC	Grap.1
F3C1-F3C2	"	"	TGC	Grap.1
F3C3-F3C4	"	"	TAS	Grap.1
F3C5-F3C6	"	"	TGS	Grap.1
F3C7-F3C8	"	"	TNC	Grap.2
F3C9-F3CA	"	"	TC	Grap.2
F3CB-F3CC	"	"	TGC	Grap.2
F3CD-F3CE	"	"	TAS	Grap.2
F3CF-F3D0	"	"	TGS	Grap.2
F3D1-F3D2	"	"	TNC	Multic.
F3D3-F3D4	"	"	TC	Multic. (inutilisée)
F3D5-F3D6	"	"	TGC	Multic.
F3D7-F3D8	"	"	TAS	Multic.
F3D9-F3DA	"	"	TGS	Multic.

F3DF	contenu	de	R0
F3E0	"	"	R1
F3E1	"	"	R2
F3E2	"	"	R3
F3E3	"	"	R4
F3E4	"	"	R5
F3E5	"	"	R6
F3E6	"	"	R7

F922-F923	adresse	courante	de la	TNC
F924-F925	"	"	" "	TGC
F926-F927	"	"	" "	TGS
F928-F929	"	"	" "	TAS
F92A-F92C				

accumulateur graphique (voir ci-dessous).

LES PORTS DU VDP

Le VDP est connecté à deux ports d'entrées sorties du Z80 :

— Le port 99H permet de communiquer avec les registres du VDP, ou de donner l'adresse dans la Vidéoram où l'on veut lire (ou écrire) : c'est le port de commande.

— Le port 98H reçoit (ou transmet) les données depuis (vers) la vidéoram : c'est le port de données.

■ *Ecriture dans un registre*

Pour écrire en BASIC une donnée d dans le registre r, il suffit de faire VDP(r)=d ; en ASSEMBLEUR, il faut écrire d sur le port 99H, puis r augmenté de 128 (80H), soit (r OR 80H), sur le même port. Le bit 7 de la deuxième écriture doit être à 1 pour savoir que c'est dans un registre qu'on veut écrire.

Une telle routine existe déjà dans la ROM : chargez B avec la donnée à écrire, C avec le numéro du registre, et faites un CALL 47H. La région de communication est mise à jour par cette routine. On peut aussi la faire soi-même :

```
LD      C,99H
LD      A,valeur
OUT     (C),A
LD      A,N° registre
OR      80H
OUT     (C),A
```

On aurait pu remplacer OR 80H par SET 7,A. Ici, on n'a pas mis à jour la RC, mais ce n'est pas difficile.

■ *Lecture d'un registre*

En réalité, seul le registre d'état peut être lu ; la fonction BASIC PRINT VDP (0 — 8) ne fait que lire des octets de la région de communication qui ont été mis à jour lors des dernières écritures dans ces registres. Pour lire le registre d'état, en BASIC comme en ASSEMBLEUR on lit le port 99H. La routine existe en ROM : faites un CALL 13EH, vous aurez le résultat dans A.

■ *Lecture de la Vidéoram*

Pour lire le contenu d'une adresse de la vidéoram, il faut :

— Décomposer l'adresse en une partie forte et une partie faible
— Ecrire la partie faible sur le port 99H
— Ecrire la partie forte sur le port 99H
— Lire la donnée sur le port 98H.

Il existe une routine en ROM pour cela : il suffit de charger HL avec l'adresse à lire et de faire un CALL 4AH.

Le résultat se trouve alors dans A.

On peut aussi faire simplement soi-même une telle routine : on suppose aussi que l'adresse à lire est dans HL :

```
LD      C,99H      ;Ecriture partie basse
OUT     (C),L      ;Sur le port 99H
OUT     (C),H      ;Ecriture partie forte
EX      (SP),HL    ;Petit délai
EX      (SP),HL    ;Petit délai
DEC     C          ;Lecture de la donnée
IN      A,(C)      ;Sur le port 98H
```

Une autre routine du BIOS positionne simplement une adresse VRAM en lecture : c'est en 50H, avec l'adresse dans HL.

■ *Ecriture dans la Vidéoram*

Pour écrire une donnée d à une adresse ad dans la Vidéoram, il faut :

— Décomposer l'adresse en une partie forte et une partie faible
— Ecrire la partie faible sur le port 99H
— Ecrire la partie haute augmentée de 64 (40H) sur le même port 99H
— Ecrire la donnée sur le port 98H.

Il existe une routine de la ROM qui fait ce travail : chargez HL avec l'adresse dans la Vidéoram, A avec la valeur à écrire, et faites un CALL 4DH.

Toutefois, il peut être intéressant de faire soi-même une telle routine.

On suppose également que HL contient l'adresse dans la Videoram et A la donnée à écrire.

```
PUSH AF          ;On sauve A
LD C,99H         ;Ecriture sur le port 99H
OUT (C),L        ;De la partie basse de l'adresse
LD A,H           ;Partie haute
OR 40H           ;Augmentée de 40H
OUT (C),A        ;Sur le port 99H
EX (SP),HL       ;Petit délai
EX (SP),HL       ;Petit délai
POP AF           ;On restaure A
DEC C            ;Ecriture sur le port 98H
OUT (C),A        ;De la donnée contenue dans A
```

Un OR 40H suffit pour augmenter la partie haute de l'adresse de 40H, car la Vidéoram ne faisant que 16Ko, cette partie haute est au maximum égale à 3FH. C'est cette particularité qui permet d'utiliser le bit 3 de la deuxième écriture sur le port 99H pour distinguer une lecture d'une écriture dans la Vidéoram ; l'écriture dans un registre est décelée par le bit 7.

Une autre routine du BIOS permet de positionner une adresse VRAM en écriture : c'est en 53H, avec l'adresse VRAM dans HL.

ATTENTION, après une lecture (ou écriture), l'adresse dans la Vidéoram s'auto-incrémente ; une nouvelle lecture du port 98H donnera le contenu de l'adresse immédiatement supérieure sans avoir besoin de fournir de nouvelle adresse au VDP. L'accès à plusieurs adresses consécutives de la Vidéoram ne demande donc qu'une seule instruction. Par contre, après que l'adresse ait été fournie au VDP, il faut un petit délai pour que la donnée à lire ou à écrire soit valide sur le port 98H. C'est le but des deux instructions EX (SP),HL dans les exemples ci-dessus (du point de vue des registres deux échanges successifs ne changent rien).

LES MODES

■ *Le mode Texte*

Fonctionnement

C'est le plus simple. Les sprites ne sont pas actifs. Deux couleurs pour tout l'écran, l'une pour « l'encre », l'autre pour le fond. Ces couleurs sont fixées par la valeur du registre 7, comme nous l'avons expliqué plus haut.

Il y a 24 rangées de 40 cellules, soit 960 positions à l'écran numérotées de 0 à 959 par rangée de 40 ; la première (cellule N° 0) est en haut à gauche de l'écran, la 40e (cellule N° 39) est en haut à droite, la 960e (cellule 959) est en bas à droite de l'écran. Chaque cellule est une matrice 6×8 points.

La Table de Noms occupe donc 960 octets ; elle contient les codes ASCII des caractères à afficher aux positions correspondantes à l'écran. Son adresse en

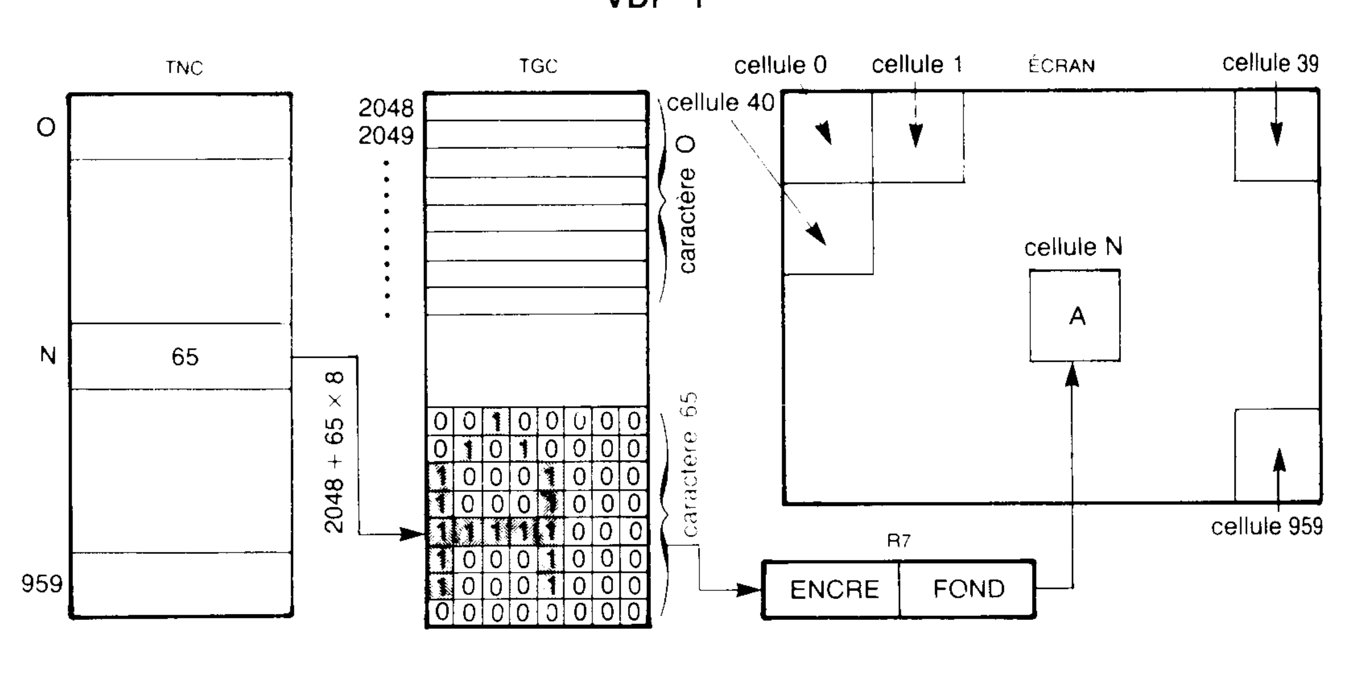
standard (à l'initialisation du mode) est 0. Pour afficher par exemple un A à la 6e cellule de la 2e ligne, il suffit de mettre 65 (soit 41H, le code ASCII de A) à l'adresse 45 de la Vidéoram.

La Table Génératrice des Configurations contient les dessins (8 octets par caractère) des 256 caractères disponibles, rangés par ordre ASCII. En mode Texte, les deux bits les moins significatifs de chaque octet de la TGC sont ignorés, ce qui donne des caractères 6×8 points (et 40 cellules par rangée). Cela explique pourquoi les caractères semi-graphiques sont « tronqués » à leur droite. Les caractères alphanumériques, eux, sont définis sur 6×8 (les deux bits de droite sont à 0), et donc apparaissent entiers en texte et « espacés » en Graphique 1, qui les affiche dans des cellules 8×8.

Les 1 de la TGC seront affichés de la couleur définie par les 4 bits les plus significatifs du registre 7, les 0 seront de la couleur définie par les 4 bits les moins significatifs de ce même registre.

L'adresse de la TGC en mode texte est 800H (2048) en standard.

A l'initialisation du mode texte, la TNC est remplie de 32 (caractères espace), ce qui donne un écran vide ; la TGC est chargée avec le jeu de caractères qui se trouve en ROM, dans le même ordre, à partir de l'adresse 1BBFH.



Récapitulation

Table	Adresse	Longueur
TNC	de 0000 à 959 (3BFH)	960 oct.
TGC	de 800H à 0FFFH (2048 à 4095)	2048 oct.
TC	inutilisée	
TGS	inutilisée	
TAS	inutilisée	

Une récapitulation générale de tous les registres dans tous les modes se trouve en fin de chapitre.

Trucs, astuces, adresses relatifs au mode texte

Pour entrer dans le mode texte, vous disposez de plusieurs moyens :

— Faire un CALL 6CH. Ceci initialise le mode texte, c'est-à-dire positionne les tables aux adresses de la RC, et charge la TGC avec le jeu de caractères.

— Faire un CALL 5FH, avec 0 (pour texte) dans A. Cette routine appelle la précédente.

— Faire un CALL 78H. Ceci FORCE le mode texte (positionne les registres R0 et R1 de façon adequate), et remet les tables aux adresses contenues dans la région de communication, mais SANS les recharger. Cela veut dire que si vous êtes passé temporairement par un autre mode qui modifie les contenus des adresses correspondant à la TGC texte, vous n'avez plus le jeu de caractères. Attention par conséquent aux manipulations dans la VRAM, elles réservent parfois des surprises.

— Ecrire 0 dans R0 et F0H dans R1, puis positionner la TNC et la TGC où vous voulez à l'aide des registres R2 et R4, mettre à jour la RC, enfin recharger le jeu de caractères depuis la ROM (il est stocké de 1BBFH à 23BEH (256*8 octets)).

Vous pouvez obtenir jusqu'à 14 pages écran différentes, simplement en changeant l'adresse de la TNC. En effet, celle-ci a 960 octets de long et vous pouvez la disposer où vous voulez par pas de 1K (1024) dans les 16K de la Vidéoram. Cependant, n'oubliez pas la place de la TGC qui fait 2K (2048 octets). Il reste donc 14 possibilités pour loger la TNC.

Pour cela, vous devez écrire les 4 bits les plus significatifs de la nouvelle adresse de la TNC dans le registre 2 ; de plus, si vous utilisez la ROM dans votre programme, vous devez mettre à jour la région de communication. Voici une routine qui passe la TNC Texte à l'adresse 2000H de la VRAM :

```
LD      A,20
LD (0F3B4H),A ;Mise a jour RC (TNC)
LD      A,8
LD (0F3E1H),A ;Mise a jour RC (R2)
LD      B,A
LD      C,2    ;Ecriture de 8
CALL    47H    ;dans R2
LD HL,(0F3B3H);Adr. nouvelle TNC
LD      BC,960 ;Longueur TNC
LD      A,' '  ;Caractere espace
CALL    56H    ;Vide l'ecran
```

Vous pouvez aussi changer un ou plusieurs caractères. Reportez-vous à la Figure VDP1 : vous voyez que chaque caractère est défini par 8 octets dans la TGC à partir

de l'adresse 2048+8×(code ASCII). Pour modifier un caractère donné, il suffit de modifier les 8 octets de la TGC qui lui correspondent.

Voici une routine qui effectue ce travail : les 8 octets qui formeront le nouveau caractère sont stockés en RAM, à partir d'une adresse quelconque ; on suppose que DE contient cette adresse, et A le code ASCII du caractère à modifier :

```
LD BC,(0F3B7H) ;Adres TGC
LD L,A         ;ASCII DANS L
XOR A          ;Annule A
LD H,A         ;Annule H
SLA L          ;
RL H           ;HL=HL*2
SLA L          ;
RL H           ;HL=HL*2
SLA L          ;
RL H           ;HL=HL*2 (HL=8*ASCII)
ADD HL,BC      ;HL=ad+TGC+8*ASCII
EX HL,DE       ;Pour la routine
LD BC,08       ;Nb d'octet a transferer
CALL 56H       ;
```

Nous avons utilisé la routine de la ROM en 56H. Celle-ci transfère en Vidéoram à l'adresse contenue dans DE la table (suite quelconque d'octets) dont l'adresse RAM est dans HL et la longueur dans BC. Cette routine sert souvent mais attention, elle modifie tous les registres.

Notez la manière de multiplier HL par 8 : on emploie trois groupes de décalages à gauche, chacun multipliant L par 2 (SLA L), et récupérant l'éventuelle retenue dans H (RL H).

Si vous voulez modifier plusieurs caractères de code ASCII consécutifs, il vous suffit de charger BC avec 8 fois le nombre de caractères à changer (à condition que la table pointée par DE contienne tous les nouveaux caractères). A contiendra le code ASCII du 1er caractère à changer.

Vous pouvez bien entendu par ce moyen changer TOUS les caractères, et obtenir un autre jeu. Plus, vous n'êtez pas obligé de charger ce nouveau jeu de caractères par dessus l'ancien ; vous pouvez créer une autre TGC, et basculer de l'une à l'autre à l'aide du registre 4. Vous avez alors 2 jeux de caractères simultanés, que vous pouvez utiliser avec des pages écran différentes (cf. plus haut) ! N'oubliez pas dans votre euphorie calligraphique, que tout risque d'être perdu si vous passez dans un autre mode au milieu de votre programme, à moins d'avoir sauvé avant vos pages écran en RAM centrale grâce à la routine symétrique en 59H (lecture d'une table de la VRAM à l'adresse contenue dans HL, de longueur contenue dans BC, et recopie en RAM à l'adresse contenue dans DE). Voici pour terminer une routine qui positionne la TNC en 3000H et la TGC en 1000H, et qui recharge celle-ci depuis la ROM avec le jeu de caractères.

```
LD       BC,0      ;
CALL     47H       ;Ecrit 0 dans R0
INC      C         ;Pour ecrire dans R1
LD       B,0F0H    ;Pour passer en texte
CALL     47H       ;Ecrit F0H ds R1
INC      C         ;Pour ecrire ds R2
LD       B,0CH     ;4 bits forts TNC
CALL     47H       ;TNC en 3000H
LD       C,4       ;Pour ecrire ds R4
LD       B,02      ;3 bits forts TGC
CALL     47H       ;TGC en 1000H
LD       A,30H     ;
LD  (0F3B4H),A     ;Adresse TNC ds RC
LD       A,10H     ;
LD  (0F3B8H),A     ;Adresse TGC ds RC
LD       A,' '     ;Caract. espace
LD  HL,(0F3B3H)    ;Adresse TNC
LD       BC,960    ;Nb cellules ecran
CALL     56H       ;Vide nouvelle TNC
LD  DE,(0F3B7H)    ;Adresse TGC
LD  HL,1BBFH       ;Adr. jeu de caract.
LD  BC,800H        ;Longueur jeu caract.
CALL     5CH       ;Transfert en VRAM
```

- *Le mode graphique 1*

Fonctionnement

En mode graphique 1 (Gr. 1), toutes les tables sont actives. Les sprites peuvent être utilisés (nous les étudierons plus loin).

L'écran est maintenant divisé en 24 rangées de 32 cellules, soit 768 positions numérotées de 0 à 767 ; la première (cellule 0) est située en haut à gauche de l'écran, la 32ᵉ (cellule n° 31) est en haut à droite, la 768ᵉ (cellule n° 767) est en bas à droite de l'écran. Chaque cellule est constituée cette fois d'une matrice 8×8 points.

Le fonctionnement est semblable à celui du mode texte : la Table de Noms des Configurations (768 octets) contient toujours les codes ASCII des caractères à afficher en positions correspondantes. Son adresse en standard est 1800H (6144), modifiable bien sûr par le registre 2.

La Table Génératrice des Configurations contient toujours les dessins des 256 caractères affichables. Son adresse standard est 0 (modifiable par le registre 4).

Cette fois, la Table des Couleurs est active ; son adresse standard est 2000H (8192). Cependant, elle n'est pas d'une grande utilité : elle contient seulement 32 octets ; le premier octet définit (comme le registre 7 en mode texte) une couleur d'encre et une couleur de fond sur les 8 premiers caractères de la TGC (caractères dont les codes ASCII vont de 0 à 7). Le deuxième octet définit les couleurs des caractères 8 à 15, et ainsi de suite, le 32ᵉ octet définissant les couleurs des caractères 247 à 255.

Cela ne permet pas une gestion aisée de la couleur. Il vaut mieux (et c'est ce que fait l'instruction SCREEN 1 du BASIC) charger toute la TC avec le même octet qui définira une même couleur d'encre et une même couleur de fond sur tous les caractères.

Le bord de l'écran est de la couleur des quatre bits les moins significatifs du registre 7.

Le fonctionnement est illustré par la figure 2. Ce mode, malgré son nom (qu'il doit à la présence possible des sprites ainsi qu'aux caractères semi-graphiques 8×8 qui sont affichés entiers) ne peut servir facilement à la production de dessins, à moins de reconfigurer tous les caractères un à un.

Récapitulation

Table	Adresse	Longueur
TNC	de 1800H à 1AFFH (6144) (6911)	768 oct.
TGC	de 0 à 17FFH (2047)	6144 oct.
TC	de 2000H à 2019 (8192) (8225)	32 oct.
TGS	à partir de 3800H (14336) ; sa longueur dépend du nombre et de la taille des sprites utilisés.	
TAS	à partir de 1B00H (6912) ; sa longueur dépend du nombre de sprites (4 octets par sprite).	

Une récapitulation générale de tous les registres pour tous les modes se trouve en fin de chapitre.

Trucs, astuces et adresses relatifs au mode graphique 1

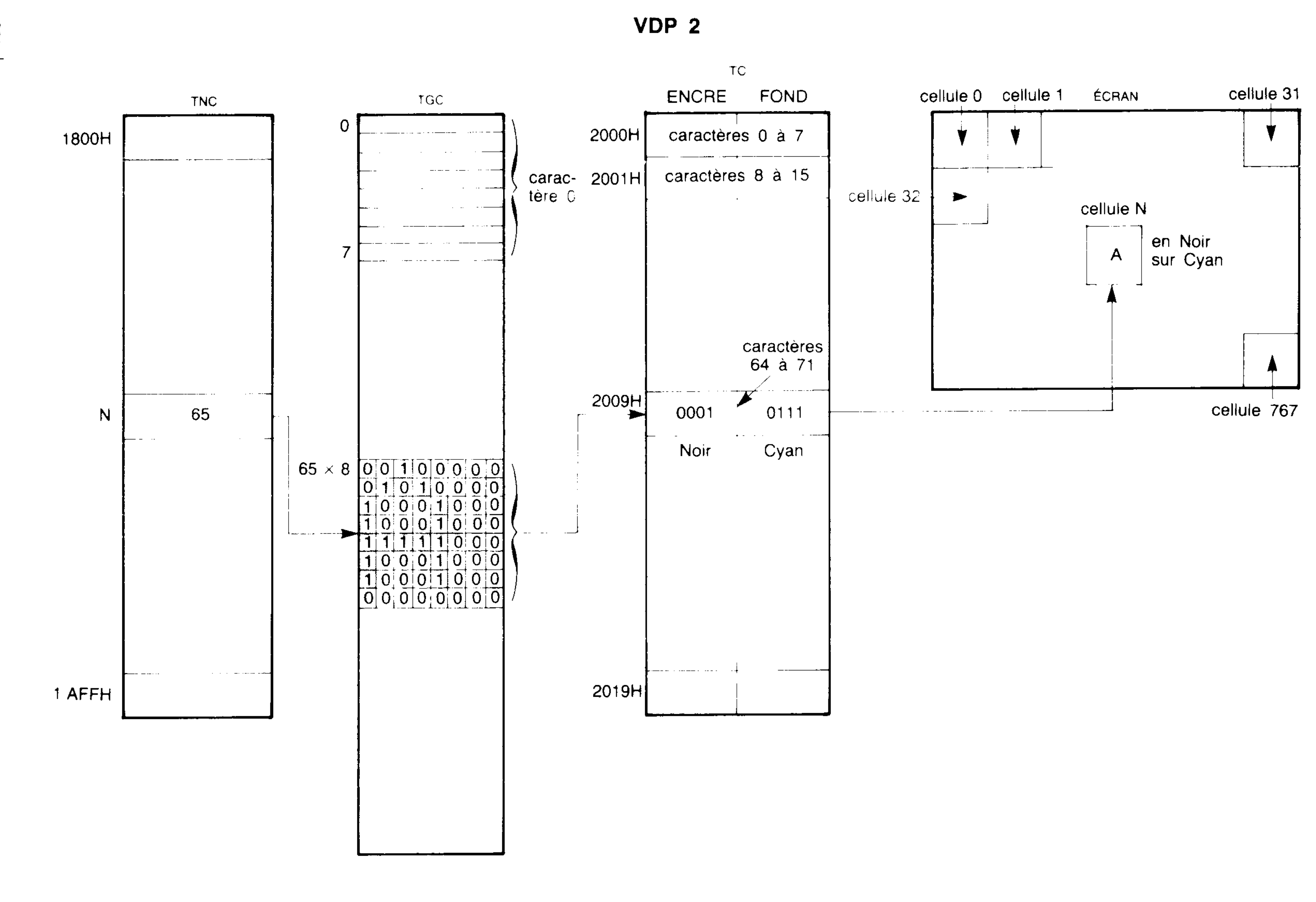
Pour entrer en GR1, vous disposez des mêmes moyens que pour entrer en mode TEXTE :

— Faire un CALL 6FH. Ceci initialise le mode (positionne les tables à leurs adresses de la RC, charge la TGC avec le jeu de caractères, remplit la TNC de 32 (espace) et la TC de F4H (blanc/bleu), et met à jour la RC).

— Faire un CALL 5FH avec 1 dans A. Cette routine appelle la précédente.

— Faire un CALL 7BH. Ceci FORCE le mode GR1 (positionne R0 et R1 de manière adéquate et remet les tables aux adresses contenues dans la région de communication (cf. paragraphe Région de communication), mais SANS les remplir.

— Ecrire 0 dans R0 et E0H dans R1. Vous pouvez ensuite positionner les tables où vous voulez à l'aide des autres registres. N'oubliez pas dans ce cas de mettre à


VDP 2
TNC
1800H
N
65
1 AFFH
TGC
0
7
carac-
tère 0
65 × 8
00100000
01010000
10001000
10001000
11111000
10001000
10001000
00000000
TC
ENCRE
FOND
2000H
caractères 0 à 7
2001H
caractères 8 à 15
caractères
64 à 71
2009H
0001
0111
Noir
Cyan
2019H
cellule 0
cellule 1
ÉCRAN
cellule 31
cellule 32
cellule N
A
en Noir
sur Cyan
cellule 767

jour la région de communication ; de cette façon, si vous quittez le mode GR1 sans altérer la Vidéoram à l'endroit de vos tables GR1, un CALL 7BH (voir ci-dessus) suffira pour y revenir.

D'une manière similaire au mode texte, vous pouvez obtenir plusieurs pages écran, et ou plusieurs jeux de caractères. Le nombre de possibilités dépend du nombre de sprites utilisés (il faut la place de la TGS et de la TAS).

Vous avez la possibilité de dessiner un fond fixe en GR1, en redessinant les caractères à votre convenance d'après les couleurs qu'ils devront prendre, et d'animer des sprites par-dessus...

■ *Le mode graphique 2 (GR2)*

C'est ici que les choses se compliquent ! Le GR2 est le mode où vous pouvez obtenir les plus beaux effets : vous pouvez dessiner point par point, mélanger les couleurs, écrire du texte, etc. Malheureusement ces possibilités se payent par une certaine complexité (ou une complexité certaine !) du fonctionnement.

Fonctionnement théorique

Théoriquement, GR2 fonctionne exactement comme GR1 à la différence près que la TGC est agrandie à 3×256 caractères, soit 6144 octets. La TNC compte toujours 768 octets, correspondant aux 768 celules 8×8 points ; on met toujours dans la TNC le nom (numéro) de la configuration à afficher, mais puisqu'il y a maintenant 3×256 configurations possibles, on ne peut plus « choisir » son numéro sur un octet. C'est pourquoi la TGC est divisée en trois parties de 256 caractères chacune et la TNC en trois parties de 256 octets (qui correspondent chacune à 1 3 de l'écran, soit 8 rangées de cellules). Chaque partie de la TNC est en relation avec la partie correspondante de la TGC, et fonctionne comme en texte ou en GR1. C'est un peu comme si on avait trois jeux de caractères différents, chacun pour un tiers de l'écran.

Pour couronner le tout, la Table des Couleurs est elle aussi agrandie, et **chaque octet** de la TGC possède un « double » dans la TC qui définit les couleurs de ses 0 et de ses 1.

Voilà ! Tout ceci représente le fonctionnement théorique de GR2. Pourquoi théorique ? GR2 peut parfaitement fonctionner de cette manière, mais il faut bien admettre que ce n'est pas très facile. Etudions maintenant comment on l'utilise.

□ *Fonctionnement pratique*

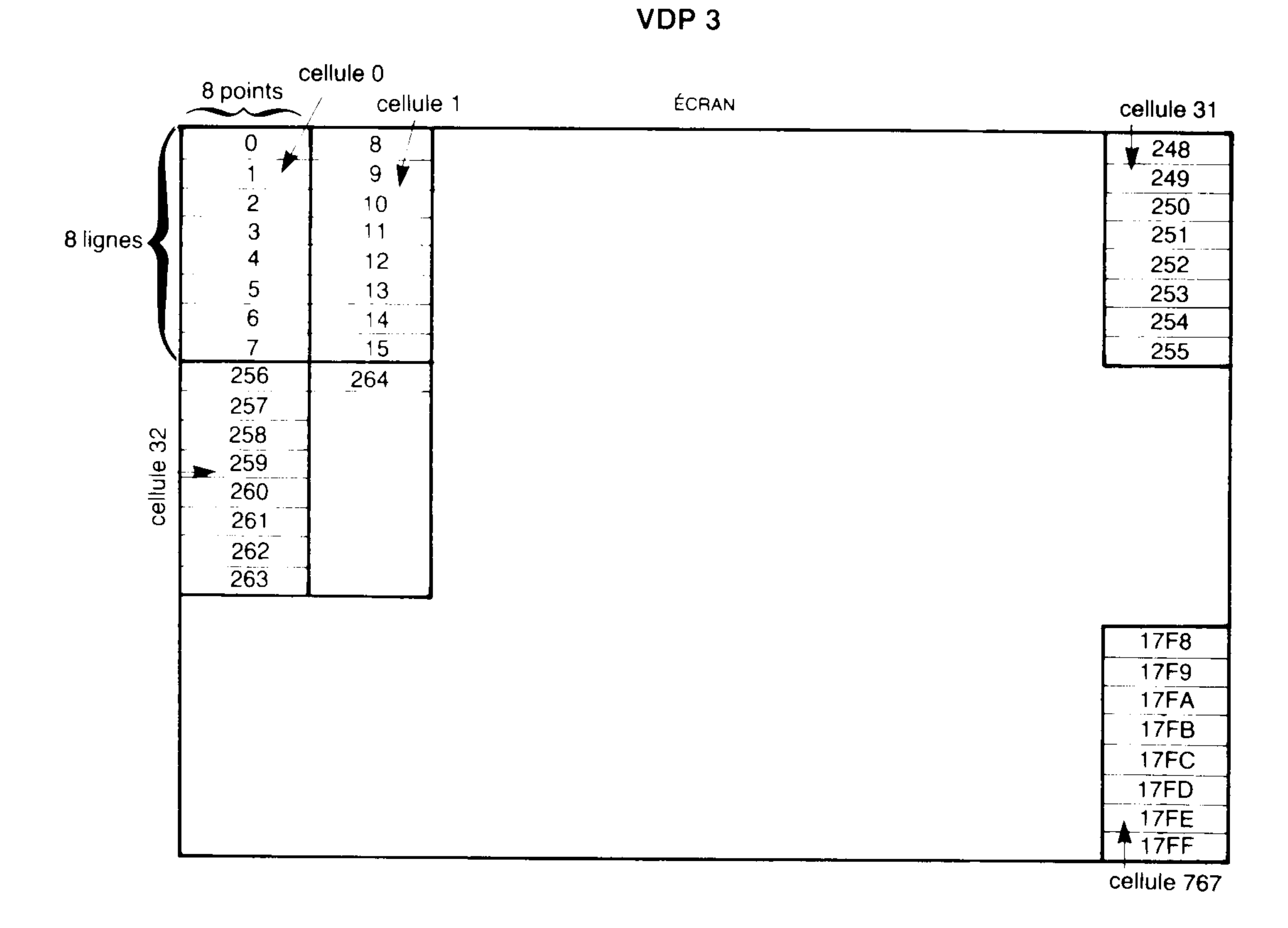
En pratique, on élimine l'intervention de la TNC. Le principe est simple : vous mettez la configuration 0 dans la cellule 0, la configuration n° 1 dans la cellule 1, ... la configuration n° 255 dans la cellule 255. On voit que pour modifier par exemple la cellule 33, il suffira de modifier la configuration 33 ; de plus, comme les trois parties de la TGC sont consécutives, il suffit de faire la même opération avec la deuxième et la troisième partie de la TNC pour que celle-ci n'intervienne plus dans la programmation : pour modifier la cellule 563, on modifiera la configuration 563.

En définitive la TNC contient trois fois la suite des entiers de 0 à 255, et on affiche directement à l'écran la configuration N dans la cellule N. Si l'on remplit la TGC de 0, on a au départ un écran vide. C'est exactement ce que fait le BASIC lors de l'instruction SCREEN 2.

La TNC éliminée, voyons comment accéder à un point particulier de l'écran.

Rapportons maintenant l'écran à un système de coordonnées habituel : en largeur, les 32 cellules donnent 256 (0 à 255) points, en hauteur les 24 rangées donnent 192 (0 à 191) lignes (souvenez-vous que les cellules sont des matrices 8×8 points). La première configuration (n 0) correspond donc aux points 0 à 7 des lignes 0 à 7 et occupe les octets 0 à 7 de la TGC ; la seconde configuration (octets 8 à 15 de la TGC) correspond aux points 8 à 15 des lignes 0 à 7, et ainsi de suite. Cela donne malheureusement une numérotation un peu fantaisiste des octets de la TGC par rapport à l'écran (cf. figure VDP 3).

VDP 3



Pour accéder à un point particulier de l'écran, il faut trouver l'adresse de l'octet de la TGC dans laquelle il se trouve, puis la place qu'il occupe dans cet octet.

La TGC est placée en standard à l'adresse 0. La formule qui donne l'adresse ad de l'octet qui contient le point de coordonnées x, y est la suivante :

$$ad = [(y - ymod8) \times 32] + (ymod8) + (x - xmod8)$$

Dans cet octet, il faut allumer le bit n 7–xmod8 (pour les récriminations, s'adresser chez Texas Instrument !).

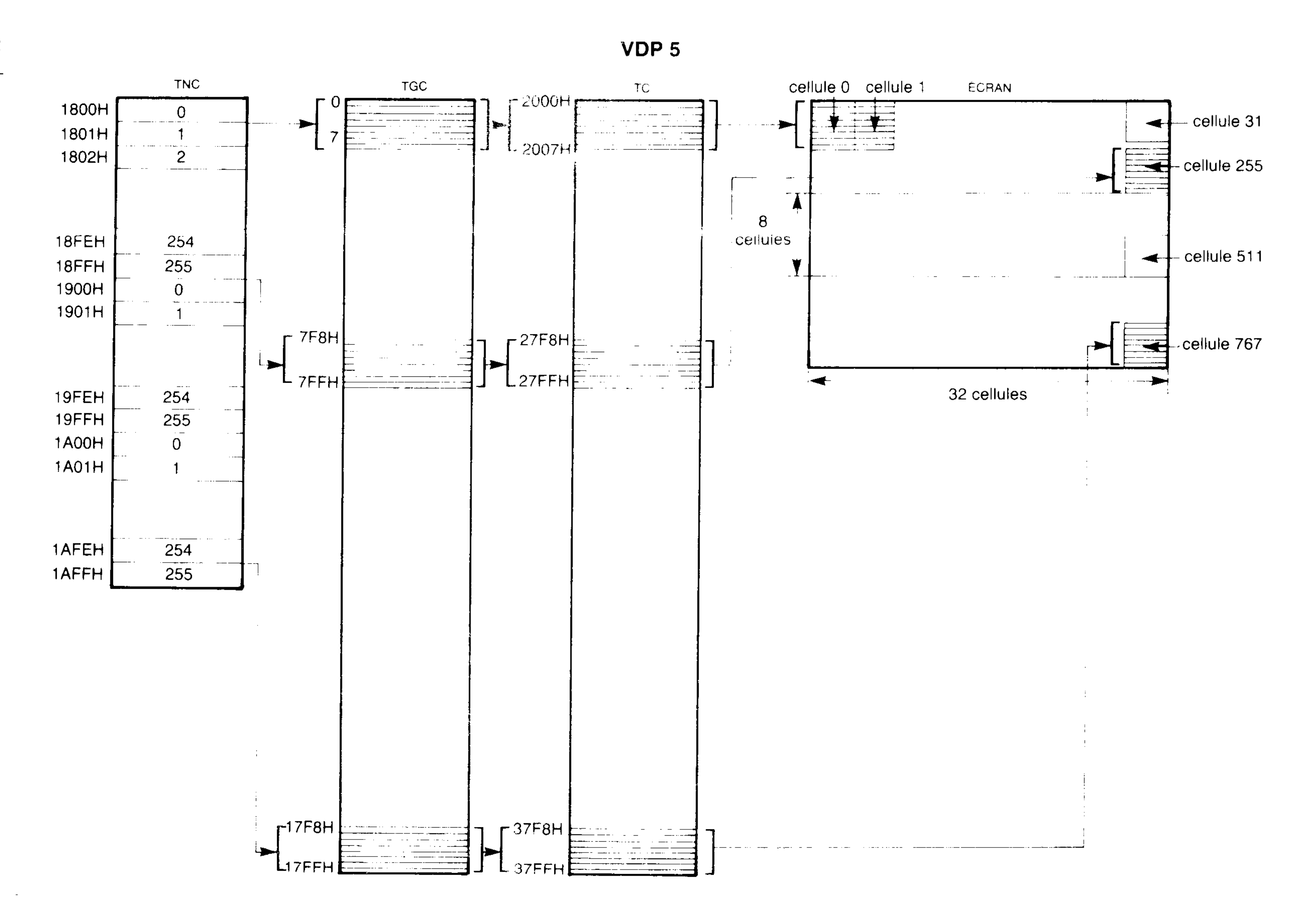
Heureusement, nous avons la ROM ! Il s'y trouve bien sûr une routine qui fait tout le travail, en 111H ; nous y reviendrons plus loin.

Nous ne démontrerons pas cette formule, mais elle peut se comprendre en regardant la figure VDP 4.



Une fois obtenue l'adresse dans la TGC (dans l'écran), il faut avoir celle dans la TC. Ce n'est pas difficile, il suffit d'augmenter la précedente de 2000H : en effet, la TC en GR2 a la même longueur que la TGC, et commence en 2000H. Chaque octet de la TGC possède un correspondant dans la TC, qui définit les couleurs de ses 0 et de ses 1.

Remarque : Vous pouvez parfaitement faire fonctionner GR1 de cette façon, mais les effets de couleurs seront beaucoup plus limités : ce n'est intéressant que si vous travaillez en deux couleurs seulement. La figure VDP5 résume le fonctionnement en GR2.


VDP 5
TNC
1800H
1801H
1802H
18FEH
18FFH
1900H
1901H
19FEH
19FFH
1A00H
1A01H
1AFEH
1AFFH
0
1
2
254
255
0
1
254
255
0
1
254
255
TGC
0
7
7F8H
7FFH
17F8H
17FFH
TC
2000H
2007H
27F8H
27FFH
37F8H
37FFH
ECRAN
cellule 0
cellule 1
cellule 31
cellule 255
8 cellules
cellule 511
cellule 767
32 cellules

Récapitulation

Table	Adresse	Longueur
TNC	de 1800H à 1AFFH (6144) (6911)	768 oct.
TGC	de 0 àC17FFH (6143)	6144 oct.
TC	de 2000 à 37FFH (8192) (14335)	6144 oct.
TGS	à partir de 3800H (14336) ; sa longueur dépend du nombre et de la taille des sprites utilisés.	
TAS	à partir de 1B00H (6912) ; sa longueur dépend du nombre de sprites (4 octets par sprite).	

Trucs, astuces, et adresses relatifs au mode graphique 2

Il existe plusieurs façons d'entrer en GR2 :

— Faire un CALL 72H (ou un CALL 5FH avec 2 dans A). Ceci initialise le mode GR2 : les registres R0 et R1 sont chargés aux valeurs adéquates, les tables sont positionnées aux adresses standard. La RC est mise à jour. La TNC est remplie avec trois fois la suite 0-FFH, comme expliqué plus haut ; la TGC est remplie de 0 (écran vide). ATTENTION LA TC EST REMPLIE DE 04, donc avec l'encre transparente. Il ne suffira pas de mettre la configuration voulue dans la TGC, il faut aussi mettre les couleurs dans les octets correspondants (+2000H), sans quoi votre dessin serait invisible ! La routine suivante vous évitera cette mésaventure : elle charge la TC avec encre verte sur fond noir (changez la ligne 10 pour d'autres couleurs) :

```
10 LD    A,21H
20 LD    HL,2000H  ;adresse TC
30 LD    BC,17FFH  ;longueur TC
40 CALL  56H       ;écriture en VRAM
```

— Faire un CALL 7EH. Ceci FORCE le mode GR2 : charge R0 et R1, et positionne les tables aux adresses contenues dans la région de communication, mais ne les remplit PAS (cf. paragraphe Région de Communication).

— Ecrire 2 dans R0 et E0 dans R1, puis positionner vos tables avec les autres registres. Cependant, ce mode est gourmand en VRAM, et vous n'avez en fait que la possibilité d'échanger la TGC et la TNC, ce qui présente peu d'intérêt. Vous avez tout juste la place, si vous n'utilisez aucun sprite, de loger une page GR2 et une page texte ensemble : il faut mettre la TNC texte à la place de la TAS et la TGC texte à la place de la TGS ; cela « rentre » tout juste ! Vous passerez de l'un à autre à l'aide de CALL 7EH et CALL 78H, si vous avez pris la précaution de mettre à jour la région de communication et de mettre la valeur 208 (D0H) dans le 1er octet de la TAS (cf. sprites).

La routine 111H

Cette routine sert à calculer l'adresse dans la Vidéoram (TGC) d'après les coordonnées x et y du point à allumer, d'après les formules ci-dessus. En entrée, BC contient x, DE contient y ; en sortie HL contient l'adresse dans la VRAM, et A contient le masque à appliquer. Expliquons-nous : si l'octet contenant le point à allumer est 0, c'est-à-dire si tous ces points sont de la couleur du fond, il suffit de mettre dans la TGC la valeur contenue dans A ; par contre, s'il y a déjà des points allumés et que vous voulez allumer en plus le point x,y, il faut lire la TGC à l'adresse (HL), faire un OR avec A, et remettre le résultat dans la TNC à l'adresse (HL).

La routine 111H utilise l'octet FCAFH, qui contient le mode courant de l'écran. Cet octet doit contenir 2 (SCREEN 2) pour que la routine fonctionne correctement. Elle écrit les valeurs trouvées dans l'accumulateur graphique défini ci-dessous.

Cette routine sert également en mode multicolore, à condition que FCAFH contienne 3 (pour SCREEN 3).

L'accumulateur graphique

Trois octets de la Région de Communication ont un rôle particulier. Deux servent à conserver l'adresse courante dans la TGC ; un sert à conserver le masque. Ces trois octets constituent l'Accumulateur graphique (Acc. Gr.) :

F92A-F92B adresse dans la VRAM
F92C masque

L'Acc. Gr. est utilisé par de nombreuses routines de la ROM (en GR2 comme en Multicolore, c'est FCAFH qui l'indique) :

— **114H** : lecture de l'Acc. Gr ; (F92A) → HL, (F92C) → A.
— **117H** : écriture de l'Acc. Gr. ; HL → (F92A), A → (F92C).
— **FCH** : déplace Acc. Gr. d'un point à droite.
— **FFH** : " " " " " à gauche.
— **102H** : " " " " " en haut.
— **108H** : " " " " " en bas.
— **105H** : idem 102H avec Carry=1 si un bord est atteint
— **10BH** : idem 108H " " " " " " "
— **120H** : écriture sur l'écran de l'Acc. grap. La couleur d'encre est dans l'octet F3F2H, mais des conflits sont possibles avec la TC ; teste le mode de l'écran (FCAF) qui doit être 2 ou 3.

En dehors du BIOS, nous avons sur notre machine :

— 16AC: idem FCH avec Carry=1 si un bord est atteint
— 16D8: idem FFH " " " " " " "
— 186C: écriture sur l'écran à l'adresse (HL) de l'octet contenu dans A, la couleur est dans F3F2.

Voici une routine qui dessine un trait vertical sur l'écran GR2, en utilisant l'accumulateur graphique.

```
        CALL   72H      ;Initialisation GR2
        LD     B,30     ;Hauteur 30 pixels
BOUCLE  PUSH   BC       ;Sauve B
        CALL   ECR      ;Appel sous prog.
        POP    BC       ;RestaureB
        DJNZ   BOUCLE   ;Point suivant
        JP     FIN      ;Sortie
ECR     CALL   10BH     ;Dep accu graph vers le bas
        CALL   120H     ;Lect accu graph et
                        ;ecriture dans l'ecran
        RET             ;Retour
```

Vous pouvez écrire du texte en GR2 : il existe une routine spécialisée en 8DH: en entrée, A contient le code ASCII du caractère à afficher, et les coordonnées X et Y (comptées en cellules, et non en points, rappelez-vous la structure de GR2) sont en FCB7 (X) et FCB9 (Y). Voici une routine qui affiche tout un texte sur l'écran :

```
        LD     A,E
        LD     (0FCB7H),A
        LD     A,D
        LD     (0FCB9H),A
        LD     HL,TEXTE
        LD     A,(HL)
BOUCLE  CALL   8DH
        INC    HL
        LD     A,(HL)
        CP     0DH
        JR     NZ,BOUCLE
        JP     FIN   ;Sortie
TEXTE   DB     'I'
        DB     'C'
        DB     'I'
        DB     0DH
```

En entrée, D contient Y, E contient X. Le texte doit se terminer par un caractère RETURN (0DH).

■ *Le mode multicolore (Mult.)*

Le mode multicolore est un mode graphique (très) basse résolution. La plus petite surface adressable possible est un carré 4×4 points. L'intérêt de ce mode est limité, et réside surtout dans le fait que vous pouvez mélanger les couleurs sans restriction de voisinage (d'où son nom).

□ *Fonctionnement*

Le fonctionnement est assez compliqué, et ressemble à celui de GR2. L'écran affiche 64×48 pavés de 4×4 points. La couleur de chaque pavé peut être l'une des 15 couleurs disponibles (plus transparent). Chaque cellule est constituée de 4 pavés

(il y a donc toujours 32×24 cellules). La couleur du bord est indépendante (registre 7) et les sprites sont actifs.

La TNC a toujours 768 octets de long, est en standard à l'adresse 800H, et pointe toujours sur la TGC. La TGC est constituée de 1536 octets, et seulement 2 octets de chaque configuration sont nécessaires pour définir les 4 couleurs possibles dans une cellule, suivant le dessin suivant :

← **8 points** →

coul. A coul. B → A B ↑ **8 points** ↓

coul. C coul. D C D

2 octets de la TGC

Deux octets sont donc nécessaires pour définir une cellule (1 octet pour les 2 couleurs du haut et un autre pour les 2 couleurs du bas).

Mais l'ennui, c'est que l'emplacement de ces 2 octets au sein des 8 que comporte une configuration dépend de la place qu'occupe la cellule dans l'écran ! Pour les cellules de la première rangée (0 à 31), les deux octets utiles sont les deux premiers de chaque groupe de 8 ; pour les cellules de la deuxième rangée (32 à 61), ce sont les troisième et quatrième octets qui définissent les couleurs, et ainsi de suite... on recommence six fois ce processus pour arriver à la dernière cellule de l'écran (merci au constructeur !).

En pratique, on utilise un artifice semblable à celui de GR2 pour éliminer l'action de la TNC : en la chargeant avec six fois la suite 0-31, on peut afficher directement dans l'écran en écrivant dans la TGC. (Remarquez que les configurations étant ici des blocs de quatre couleurs, l'information couleur vient en mode Mult. de la TGC ; La TC n'est pas active dans ce mode.) Ceci produit également un numérotage des octets de la TGC sur l'écran assez compliqué (cf. figure VDP 6).

Chaque octet de la TGC définit un bloc bicolore de deux pavés (4×8 points), selon le procédé habituel des couleurs (cf. registre 7 pour le codage).

La TGC commence en standard à l'adresse 0. La formule qui donne l'adresse d'un pavé de coordonnées x,y (x de 0 à 63, y de 0 à 47) est la suivante :

$$ad = [(y - ymod8) \times 32] + [(x - xmod2) \times 4] + ymod8.$$

Si x est pair, ce sera le quartet de gauche (fort) de cet octet, sinon ce sera le quartet de droite qui définira le pavée x,y.

VDP 6

	0 1	2 3	62 63
0	0	8	248
1	1	9	249
2	2	10	250
3	3	11	251
4	4	12	252
5	5	13	253
6	6	14	254
7	7	15	255
8	256	264	
9	257	265	
40	1280		1528
41	1281		1529
42	1282		1530
43	1283		1531
44	1284		
45	1285		1533
46	1286		1534
47	1287		1535

□ *Récapitulation*

Table	Adresse	Longueur
TNC	de 800H à AFFH (2058) (2815)	768 oct.
TGC	de 0 à 5FFH (1535)	1536 oct.

Les Tables des Sprites sont exactement aux mêmes adresses qu'en GR1 ou en GR2.

□ *Trucs, astuces et adresses relatifs au mode Multicolore*

Pour entrer en Mult., vous pouvez :

— Faire un CALL 75H pour initialiser le mode (positionnement des tables à leurs adresses standard, mise à jour de la Région de communication, chargement de la TNC avec six fois la suite 0–31 comme indiqué plus haut, et remplissage de la TGC avec 0 (transparent).

— Faire un CALL 81H. Ceci FORCE le mode Mult : positionne R0 et R1 aux valeurs adéquates et remet les tables aux adresses contenues dans la Région de communication, mais SANS les remplir. Ne positionne PAS FCAFH (mode de l'écran), il faut le faire soi-même.

— Ecrire 0 dans R0 et E8H dans R1, puis positionner vos tables à l'aide des autres registres. Ce mode n'étant pas très gourmand en mémoire, vous pouvez obtenir des pages Mult. et d'autres modes ensemble dans la VRAM.

L'ensemble des routines et adresses données pour l'accumulateur graphique au paragraphe sur le mode GR2 sont utilisables si l'octet mode d'écran (FCAFH) contient 3.

LES SPRITES

Les sprites ou lutins sont actifs en mode graphique 1 et 2, ainsi qu'en multicolore. Ce sont des dessins 8×8 points (agrandissables à 16×16) ou 16×16 points (agrandissables à 32×32). Dans le cas de l'agrandissement, chaque pixel du dessin agrandit à 2×2 points, ce qui entraîne des dessins beaucoup plus grossiers.

Ce sont les bits 0 et 1 de R1 qui définissent la taille et l'agrandissement des sprites (tous les sprites ont la même taille et le même agrandissement) :

b0	b1	surface	résolution	Nb octets TGS
0	0	8×8	1 point	8
1	0	16×16	1 point	32
0	1	16×16	2×2 points	8
1	1	32×32	2×2 points	32

Le VDP peut afficher 32 sprites, sur 32 plans. Les plans ont un ordre de priorité d'affichage : en cas de superposition, le sprite N° 0 passera « devant » tous les autres ; le sprite N° 10 passera devant le N° 15 ou le N° 27, mais « derrière » le sprite N° 3 ou 8 (par exemple).

Cependant, il ne peut y avoir plus de quatre sprites sur une même ligne horizontale ; si cette règle est violée, les quatre sprites de priorités les plus grandes (N° les plus bas) sont affichés normalement. Le cinquième (et les suivants) ne sont pas affichés sur cette ligne ; de plus, le bit 6 du registre d'état est mis à 1, et le N° du cinquième sprite intrus est chargé dans les quatre bits bas du registre d'état.

Chaque sprite est défini à l'aide de deux tables, la Table de Génératrice des Sprites (TGS) et la Table des Attributs des Sprites (TAS).

- *La TGS*

La TGS contient les dessins des sprites. Elle est constituée au maximum de 2Ko (2048 octets), soit 32 groupes de 4 blocs de 8 octets. Chaque groupe définit le dessin d'un sprite. Les sprites 8×8 sont définis dans le premier bloc de 8 octets, les 3 blocs suivants sont ignorés. Les sprites 16×16 sont définis de la manière suivante : le premier bloc définit le quart supérieur gauche, le second le quart inférieur gauche, le 3e le quart supérieur droit, le 4e le quart inférieur droit. Dans tous les cas, les 1 seront affichés de la couleur définie pour chaque sprite dans la TAS, les 0 seront transparents. Ceci signifie que les sprites sont monochromes, et que vous verrez le fond (ou un sprite moins

prioritaire, s'il y en a un derrière) à travers les « trous » (0) de votre sprite. Pour éviter cela, et obtenir un dessin bicolore, il faut définir deux sprites par dessin, un par couleur, et les afficher au même endroit (mais attention au nombre de sprites sur la même horizontale). Le reste du plan non occupé par le sprite est transparent, ce qui laisse voir les plans moins prioritaires.

L'adresse de la TGS est donnée par le registre 6 (cf. Registres du VDP). Elle est en standard en 3800H (14336) pour tous les modes où elle est active.

■ *La TAS*

La table des attributs des Sprites définit la position et la couleur de chaque sprite. Elle est constituée de 32 × 4 (= 128) octets. Il y a 4 octets par sprite :

Octet 0 : coordonnée verticale
Octet 1 : horizontale
Octet 2 : nom (numéro)
Octet 4 : b 0 0 0 couleur

Le premier octet définit le **deplacement** (c'est donc une valeur signée) depuis le haut de l'écran, en points (pixels), du coin superieur gauche du sprite. ATTENTION, ce système est fait de telle façon qu'une valeur de − 1 collera le sprite contre le bord superieur de l'écran.

Si les coordonnées sont telles que le sprite est partiellement (ou entièrement) en dehors de la zone d'affichage, seule la partie visible sera affichée, le reste sera caché par le bord : ceci permet de faire venir un sprite de derrière le bord de l'ecran. Donc pour faire disparaître un sprite par le haut sous le bord, il faut lui donner une coordonnée verticale de plus en plus négative (− 2 (FDH), − 3 (FDH), etc. cf. chapitre 3) jusqu'à ce qu'il ait complètement disparu. Une coordonnee de − 31 cachera entierement un sprite 32 × 32 sous le bord superieur de l'ecran.

De la même manière, une coordonnée verticale supérieure à 191 fera disparaître le sprite par le bas (puisque la coordonnee définit le coin en haut à gauche du sprite).

Le deuxième octet définit la distance depuis le bord gauche de l'écran : une valeur de 0 collera le sprite au bord gauche. Faire disparaître un sprite à droite est donc simple : une valeur de 255 met tout le sprite sous le bord droit, des valeurs un peu plus petites le feront apparaître progressivement : le sprite sera collé au bord droit pour une valeur de 255 − 8 = 247 (sprite 8 × 8), 255 − 16 = 239 (sprite 16 × 16), ou 255 − 32 = 223 (sprite 32 × 32).

Le faire disparaître à gauche est un peu plus compliqué : les coordonnées horizontales allant de 0 à 255, on ne peut pas utiliser de valeurs signées : on emploie le bit 7 du quatrième octet. Si ce bit (on l'appelle bit de retard) est à 0, il n'a pas d'action : s'il est à 1, la coordonnée horizontale donnée par le deuxième octet est diminuée de 32. Ceci permet de cacher le sprite sous le bord gauche. N'oubliez pas, si vous déplacez ensuite ce sprite jusqu'au bord droit que lorsque vous remettrez le bit de retard à 0, il faudra augmenter la coordonnée horizontale de 32 pour le garder à la même place (ou de 31 pour le faire avancer d'une position à droite).

Le troisième octet contient le Nom, c'est-à-dire le numéro du sprite. C'est lui qui permet au VDP d'aller chercher la bonne configuration dans la TGS pour un sprite donné.

Le quatrième octet contient dans ses bits 0 à 3 la couleur du sprite (couleur des bits à 1 dans la TGS). Le codage est le même que dans la TC. Il est donné au paragraphe « Registre 7 ». Les bits 4 à 6 sont à 0. Le bit 7 est le bit de retard défini ci-dessus.

Le VDP indique que 2 sprites ou plus ont 1 point commun en mettant à 1 le bit 5 du registre d'état (Flag de coïncidence). Cependant, il ne vous donne pas les N° de ces sprites.

Le VDP scrute en permanence la TAS pour afficher les bons sprites aux bons endroits. S'il rencontre une valeur de 208 (D0H) dans la coordonnée verticale, ce processus est interrompu, et tous les sprites suivants sont ignorés. Cela signifie que vous pouvez empêcher l'affichage de tous les sprites en mettant 208 dans le premier octet de la TAS, et utiliser la place de la TGS et de la TAS pour autre chose.

■ *Routines utiles*

Il n'y a que trois routines relatives aux sprites facilement utilisables :

— **84H:** Donne l'adresse dans la TAS du sprite dont le N° est dans A. Les octets F926 et F927 (adresse courante de la TAS) doivent être à jour.

— **87H:** Idem pour la TGS. F928 et F929 doivent être à jour.

— **8AH:** Retourne dans A la taille courante des sprites (8 ou 32) d'après le registre 1, et dans Carry le bit de taille.

Les Sprites se déplacent facilement en modifiant leurs coordonnées dans la TAS.

Voici un programme qui déplace le sprite dont le numéro est dans A en haut si B = 0, à droite si B = 1, en bas si B = 2, à gauche si B = 3.

```
        CALL   87H     ;Cherche adr sprite ds TAS
        RRC    B       ;B pair ?
        JR     C,HORIZ ;Si oui depla. horiz.
        RRC    B       ;B etait-il 2 ?
        JR     C,BAS   ;Si oui depla. bas.
        CALL   4AH     ;Lire coord. vert.
        DEC    A       ;La remonter d'1 ligne
        JR     SORTIE  ;Et sortir
BAS     CALL   4AH     ;Lire coord. vert.
        INC    A       ;La descend. d'1 ligne
        JR     SORTIE  ;Et sortir
HORIZ   INC    HL      ;Adr. coord. horiz.
        RRC    B       ;B etait-il a 3 ?
        JR     C,GAUCH ;Si oui dep. gauche
        CALL   4AH     ;Sinon lire coord. horiz.
        INC    A       ;La deplacer a droite
        JR     SORTIE  ;Et sortir
GAUCH   CALL   4AH     ;Lire coord. horiz.
        DEC    A       ;La deplacer a gauche
SORTIE  CALL   4DH     ;ecrit. coord. ds TAS.
        RET            ;Fin
```

Pour finir, nous vous donnons trois récapitulatifs concernant le VDP :

Contenus des divers registres dans tous les modes (à l'initialisation)

Registre et fonction	TEXTE	GRAPHIQUE 1	GRAPHIQUE 2	MULTICOLORE
0	0	0	00000010 (02H)	0
1	11110000 (F0H)	11100000 (E0H)	11100000 (E0H)	11101000 (E8H)
2 TNC	0	00000110 (06H)	00000110 (06H)	00000010 (02H)
3 TC	0	10000000 (80H)	11111111 (FFH)	0
4 TGC	00000001	0	00000011 (03H)	0
5 TAS	0	00110110 (36H)	00110110 (36H)	00110110 (36H)
6 TGS	0	00000111 (07H)	00000111 (07H)	00000111 (07H)
7 couleur	11110100 (F4H)	00000100 (04H)	00000100 (04H)	00000100 (04H)

Table des registres du VDP

Registre \ bit	B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	0	0	0	M3	EV
1	1	BLANC	IE	M1	M2	0	TAILLE	AGRANDISSEMENT
2	0	0	0	0	4 bits de poids forts de l'adresse de la TNC			
3	8 bits de poids forts de l'adresse de la TC							
4	0	0	0	0	0	3 bits de poids forts de l'adresse de la TGC		
5	0	7 bits de poids forts de l'adresse de la TAS						
6	0	0	0	0	0	3 bits de poids forts de l'adresse de la TGS		
7	couleur du texte				couleur du fond			
ETAT	F	5 Sprt.	Coïnc.	Numéro du 5e Sprite sur l'horizontale				

Adresses standard des tables dans les divers modes

TABLES / MODES	TNC	TGC	TC	TAS	TGS
TEXTE	0000 H	0800 H	—	—	—
GRAPHIQUES 1 et 2	1800 H	0000 H	2000 H	1B00 H	3800 H
MULTICOLORE	0800 H	0000 H	—	1B00 H	3800 H

10

Le circuit d'entrée/sortie : le PPI

Nous avons déjà mentionné ce circuit dans le chapitre 8 ; nous l'avions alors appelé le PPI ; pour plus de commodité, nous continuerons à le faire.

Le PPI est un circuit d'entrée sortie fabriqué sous la référence 8255 par la firme Intel. Dans le MSX, c'est lui qui se charge de la gestion des Slots (cf. chapitre 8) et de celle du clavier. Si la première partie de ce chapitre (consacrée aux slots) n'est pas fondamentale, la deuxième, quant à elle, est très importante puisqu'elle concerne le clavier, périphérique sans lequel on ne peut pas faire grand chose.

Le PPI possède 3 ports (sur 8 bits) que l'on a pour coutume de nommer A, B et C. Ces ports peuvent être utilisés selon plusieurs modes. Nous n'en étudierons qu'un seul car il est suffisant pour les MSX.

LES FONCTIONS DES PORTS

- *Port A*

Le port A sert à la gestion des slots. Nous avons déjà vu que la mémoire était divisée en quatre banks et que chaque bank avait le choix entre 4 slots, ce qui permettait notamment d'étendre la capacité mémoire des MSX à plus de 64 Ko. Comme il y a 4 banks, il était logique de diviser les 8 bits du port A en 4 fois 2 bits, chaque paquet de deux bits étant réservé pour un bank donné. Ainsi on a :

Bits 0 et 1 réservés pour le bank 0
Bits 2 et 3 réservés pour le bank 1

Bits 4 et 5 réservés pour le bank 2
Bits 6 et 7 réservés pour le bank 3

Avec deux bits on a 4 possibilités de valeur qui sont : 00, 01, 10 et 11

Ceci explique pourquoi un bank a le choix entre 4 slots. Dans la configuration standard (donnée au chapitre 8) ce sont les 4 slots 0 qui sont sélectionnés. Les autres pourront éventuellement l'être si vous possédez des extensions de mémoire.

- *Port B*

Le port B sert à la lecture du clavier. On envoie le numéro de la ligne clavier à scruter par le port C et on récupère le résultat (les touches enfoncées) dans le port B. Nous verrons cela en détail dans la deuxième partie de ce chapitre.

- *Port C*

Le port C est divisé en deux parties égales de 4 bits, la partie haute (b4 à b7) et la partie basse (b0 à b3). La partie basse est à utiliser conjointement avec le port B lors de la lecture du clavier que nous verrons plus loin. La partie haute possède 4 fonctions (une par bit) dont voici le détail.

Bit 4 : bascule de démarrage/arrêt du magnétophone.
Bit 5 : bascule d'écriture sur le magnétophone.
Bit 6 : bascule allumer/éteindre la lampe témoin du mode majuscule
Bit 7 : sert au signal SOUND

- *Utilisation des ports du PPI*

La communication du PPI avec le Z80 est assurée par des ports d'entrée/sortie dont voici le détail :

adresse	**fonction**
A8H	Ecr/Lect du port A
A9H	Lect du port B
AAH	Ecr/Lect du port C
ABH	Ecr du registre de contrôle

Voyons d'abord comment fonctionne le registre de contrôle. Ce registre peut fonctionner selon deux modes qui sont : le mode de positionnement de bits et le mode de contrôle. C'est le bit 7 de ce même registre qui détermine son mode de fonctionnement.

- *Mode positionnement de bits*

Pour ce faire, le bit 7 de ce même registre doit être à 0. Ce registre permet alors de positionner un bit donné du port C à une valeur voulue. On a alors les fonctions suivantes affectées aux divers bits :

Bit 7 : à 0 obligatoirement
Bits 6,5,4 : inutilisés
Bits 3,2,1 : donne le numéro du bit à positionner (Ce numéro est compris entre 0 et 7 et permet donc de positionner n'importe quel bit.)
Bit 0 : donne la valeur du bit à positionner (0 signifie qu'il faut mettre le bit correspondant à 0 et 1 qu'il faut le mettre à 1.)

- *Mode contrôle*

Le registre permet alors de contrôler les ports A, B et C ainsi que le mode de fonctionnement. On a alors les fonctions suivantes affectées aux divers bits :

Bit 7 : à 1 obligatoirement
Bit 6 et 5 : sélection du mode de fonctionnement du PPI. Nous vous avons dit que le PPI possédait 3 modes de fonctionnement. Nous n'étudions que le premier car c'est le seul qui sert sur les MSX. Pour être dans ce mode (dit MODE 0), les bits 6 et 5 doivent être tous deux mis à 0.
Bit 4 : sens de fonctionnement du port A. 0 signifie qu'il fonctionne en sortie, 1 qu'il fonctionne en entrée.
Bit 3 : sens de fonctionnement de la partie haute du port C (bit b4 à b7). 0 signifie que cette partie fonctionne en sortie, 1 qu'elle fonctionne en entrée.
Bit 2 : fonction analogue à celle des bits 6 et 5. Sera toujours à 0 dans le MSX.
Bit 1 : sens de fonctionnement du port B. 0 signifie qu'il fonctionne en sortie et 1 qu'il fonctionne en entrée. Etant donné la fonction du port B dans les MSX, ce bit sera toujours à 1.
Bit 0 : sens de fonctionnement de la partie basse du port C. 0 signifie que cette partie fonctionne en sortie et 1 qu'elle fonctionne en entrée. Etant donné la fonction de la partie basse du port C dans les MSX, ce bit sera toujours à 0.

Après avoir mis une valeur correcte dans ce registre de contrôle, nous allons voir comment on se sert des ports B et C pour scruter notre clavier.

Nous avons dit que pour scruter le clavier nous avions besoin de la ligne à scruter dans la partie basse du port C. En effet, dans les MSX, le clavier est divise en 9 lignes dont nous donnons le détail dans le prochain tableau. Sur chacune de ces lignes, 8 touches ont été implantées. Si l'on veut scruter une ligne, il faut donc transmettre le numéro de la ligne dans la partie basse du port C, et l'on récupère en lisant le port B l'ensemble des touches enfoncées sur cette ligne avec la convention suivante :

Un bit à 0 signifie que la touche est enfoncée
Un bit à 1 signifie que la touche n'est pas enfoncée.

Exemple : vous voulez voir si la touche A est enfoncée. Vous pouvez alors faire le petit programme suivant :

```
LD A,2              ; Accu := numéro de ligne
OUT (0AAH),A        ; Envoie ligne sur port C
IN (0A9H),A         ; Lecture port B
AND 2               ; Masquer les autres bits
JP NZ,Traitement    ; Traitement de la touche A
```

Ce principe de scrutation de clavier est très pratique car il permet de voir si plusieurs touches sont enfoncées simultanément. On peut donc lors de l'élaboration d'un jeu d'animation se déplacer tout en tirant, ou faire des choses analogues.

Voici le tableau matriciel de codage du clavier :

Tableau de codage du clavier

Port B / Port C	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Ligne 0	7	6	5	4	3	2	1	0
Ligne 1	;	[	]		+		9	8
Ligne 2	B	A	Acc	?	>	<	£	"
Ligne 3	J	I	H	G	F	E	D	C
Ligne 4	R	Q	P	O	N	M	L	K
Ligne 5	Z	Y	X	W	V	U	T	S
Ligne 6	F3	F2	F1	CODE	CAP(↓)	GRAPH	CTRL	SHIFT
Ligne 7	RETURN	SELECT	BS	STOP	TAB(→)	ESC	F5	F4
Ligne 8	→	↓	↑	←	DEL	INS	CLS	SPACE

Voici à présent quelques routines présentes en ROM servant au contrôle des diverses parties du PPI.

138H : Ce crochet appelle en ROM une routine se chargeant de la lecture du port A du PPI. Le seul paramètre de retour est la valeur contenue sur ce port qui se trouve alors dans l'accumulateur. Seul l'accumulateur est modifié lors de l'appel de cette routine.

13BH : Ce crochet appelle en ROM une routine se chargeant de l'écriture d'une donnée dans le port A du PPI. La valeur que l'on veut mettre sur le port A doit se trouver dans l'accumulateur. Aucun registre n'est modifié par l'appel de cette fonction.

141H : Sans doute la routine la plus utile : Elle permet la scrutation directe du clavier. Le paramètre d'entrée est le numéro de la ligne à scruter qui doit se trouver dans l'accumulateur. Le paramètre de retour est l'état de la ligne demandée (les touches

enfoncées correspondent à des bits à 0) qui se trouve dans l'accumulateur. Les registres modifiés par l'appel de cette routine sont A et F.

9FH : Réalise une scrutation clavier et retourne dans A le code ASCII de la touche enfoncée s'il y en a une, met Z à 1 sinon. La routine suivante réalise l'attente de la pression d'une touche :

```
ATT   CALL  9FH
      JR    Z,ATT
      RET
```

11

Le générateur de sons : le PSG

Votre MSX possède un générateur de sons de référence AY-3-8910 fabriqué par la firme General Instrument. Ce générateur de son a le nom générique de PSG (pour *Programmable Sound Generator*). Il s'agit d'un processeur très puissant permettant toutes les fantaisies musicales imaginables. Il est à noter que celui-ci ne « vole » pas de temps au système, si ce n'est pour la transmission des paramètres nécessaires à son fonctionnement. En effet, une fois que le PSG a reçu un certain nombre de paramètres (fréquence, volume, enveloppe, etc.), il exécute sa tâche en laissant le Z80 libre de faire autre chose. On peut ainsi réaliser deux tâches simultanément (par exemple déplacer un objet en jouant un son donné). Cependant, chaque fois que l'on devra transmettre de nouveaux paramètres au PSG, il faudra ne plus se consacrer qu'à lui. Ceci diffère sensiblement des systèmes où il n'y a pas de PSG et où l'on ne peut jouer de la musique que grâce au microprocesseur. Dans ces systèmes, il est possible de jouer de la musique, mais pendant qu'on la joue, on ne peut rien faire d'autre.

Notre PSG, quant à lui possède :

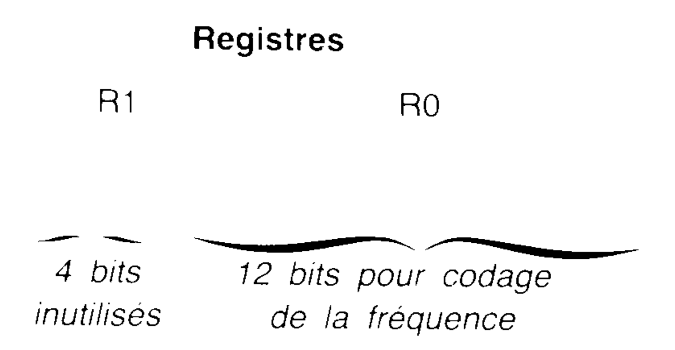
- 3 canaux indépendants réglables chacun en fréquence et en volume, il est donc très simple d'obtenir des accords.
- Un générateur de bruit blanc
- Une unité permettant de mélanger les diverses voies
- Un générateur d'enveloppe
- 2 ports d'entrées/sorties

Ces diverses unités sont accessibles par l'intermédiaire de registres internes au PSG. Ces registres sont au nombre de 16 (numérotés de R0 à R15) et ont bien sûr tous 8 bits. Nous allons les étudier un à un avant de voir comment l'on programme le PSG.

■ *Registres R0-R1, R2-R3 et R4-R5*

On se doit de rassembler ces registres par paires car chaque paire a la même fonction mais pour un canal différent. Ainsi la paire R0-R1 concerne le canal A, la paire R2-R3, le canal B et la paire R4-R5, le canal C.

Chaque paire de registre sert à régler la fréquence produite dans le canal correspondant. Cette fréquence et codée sur 12 bits. Il y aura donc 4 bits inutilisés par paire de registres. Voyons à présent le codage de cette fréquence. Les bits sur lesquels elle se code sont les 8 bits du premier registre de la paire et les 4 bits faibles du second registre de la paire. La fréquence est codée avec le second registre comme octet de poids fort et le premier comme octet de poids faible. Ainsi, pour le canal A on aura :



Puisque le codage de la fréquence se fait sur 12 bits, un nom représentant une fréquence est nécessairement compris entre 1 et 4095. Voyons maintenant comment l'on traduit une fréquence donnée en un nombre compris entre 0 et 4095.

La valeur de ce nombre est donnée par la formule :

$$Valeur = Arrondi \left(\frac{3579545}{16 \times \text{Fréquence}} \right)$$

Calculons la valeur correspondant à une note : soit le do moyen ayant pour fréquence 523,25 Hz.

$$Valeur = Arrondi \left(\frac{3579545}{16 \times 523{,}25} \right)$$

Valeur = 428 00011010100 en binaire

Pour obtenir cette fréquence sur le canal A, il faut donc mettre 1 dans R1 et ACH (soit 172 en décimal) dans R0.

Voici un tableau fournissant les fréquences des notes sur plusieurs octaves :

Tableau de fréquences

Note	**Fréquence**	**Note**	**Fréquence**
Do	130.810	Do	523.250
Ré	146.830	Ré	587.330
Mi	164.810	Mi	659.260
Fa	174.610	Fa	698.460
Sol	196.000	Sol	783.990
La	220.000	La	880.000
Si	246.940	Si	987.770
Do	261.630	Do	1046.500
Ré	293.660	Ré	1174.700
Mi	329.630	Mi	1318.500
Fa	349.230	Fa	1396.900
Sol	392.000	Sol	1568.000
La*	440.000	La	1760.000
Si	493.880	Si	1975.500

* *(Diapason)*

- *Registre R6*

Ce registre est utilisé pour coder la fréquence du générateur de bruit. Dans ce registre, seuls les 5 bits les moins significatifs sont utilisés. La formule utilisée pour obtenir la valeur en fonction de la fréquence est la même que pour les registres R0 à R5. Cette valeur est comprise entre 1 et 31 du fait qu'il n'y a que 5 bits. La plage de variation de la fréquence se trouve donc plus réduite.

Soit un bruit de 10000 Hz à envoyer dans le générateur de bruit. Calculons la valeur à mettre dans R6 qui correspond à cette fréquence.

Valeur = 3579545 (16×10000)
Valeur = 22 = 10110 en binaire

La valeur à mettre dans le registre R6 est donc 22.

- *Registre R7*

Le registre R7 est un registre de contrôle qui permet de déterminer ce que l'on va envoyer sur chacun des trois canaux (son ou bruit). Il est également utilisé pour les entrées sorties (cf. registres R14-R15).

Les bits b0, b1 et b2 sont utilisés pour déterminer si les canaux A, B et C sont ouverts pour les sons. Ainsi, si le bit b0 est à 0, le canal A est ouvert pour la sortie du son (s'il est à 1, le canal A est fermé pour la sortie de son). Les bits b1 et b2 servent à la même chose mais respectivement pour les canaux B et C.

Les bits b3, b4, b5 sont utilisés pour déterminer si les canaux A, B et C sont ouverts pour le bruit. Ainsi le bit b3 à 0 indique que le canal A est ouvert pour le bruit (ce même bit à 1 indiquerait qu'il est fermé). Les bits b4 et b5 servent à la même chose mais respectivement pour les canaux B et C.

Les bits b6 et b7 servent pour les entrées sorties (cf. plus loin registres R14-R15). Le bit b6 sert pour le port A et le bit b7 pour le port B. Un bit à 1 indique que le port correspondant est dirigé en entrée.

Par exemple, on veut obtenir la configuration suivante :

— Ports A et B dirigés en entrée
— Bruit présent sur les canaux A et C
— Son présent sur les canaux A et B

La valeur à mettre dans le registre R7 est alors :

0 0	1 0 1	0 1 1
port A et B en entrée	*Bruit présent sur canaux C et A*	*son présent sur canaux A et B*

La valeur à mettre dans le registre R7 est donc 2BH ou 43 en décimal.

■ *Registres R8, R9 et R10*

Ces trois registres servent à contrôler les amplitudes respectives des canaux A, B et C. Seuls les 4 bits de plus faible poids sont utilisés pour le contrôle de l'amplitude du son sortant sur un canal. Cette amplitude peut donc varier de 0 (volume minimum) à 15 (volume maximum). Le bit 4 de chacun de ces registres sert à déterminer si le canal correspondant conserve une amplitude constante dans le temps où si celle-ci varie grâce au générateur d'enveloppe. Si ce bit est à 0, l'amplitude ne varie pas dans le temps, s'il est à 1, l'amplitude est modulée par l'enveloppe (voir registres R11, R12 et R13).

Exemple : R8 = ???01010
R9 = ???00001
R10 = ???11111

Indique — que le volume du canal A est 10 (1010 en binaire) et que ce volume ne varie pas dans le temps.
— que le volume du canal B est à 1 et ne varie pas dans le temps.
— que le volume du canal C est à 15 (maximum) et qu'il est modulé par le générateur d'enveloppe (bit 4 à 1).

■ *Registres R11 et R12*

Ces deux registres sont utilisés pour le codage de la période de l'enveloppe. La valeur correspondant à une période donnée est donnée par la formule :

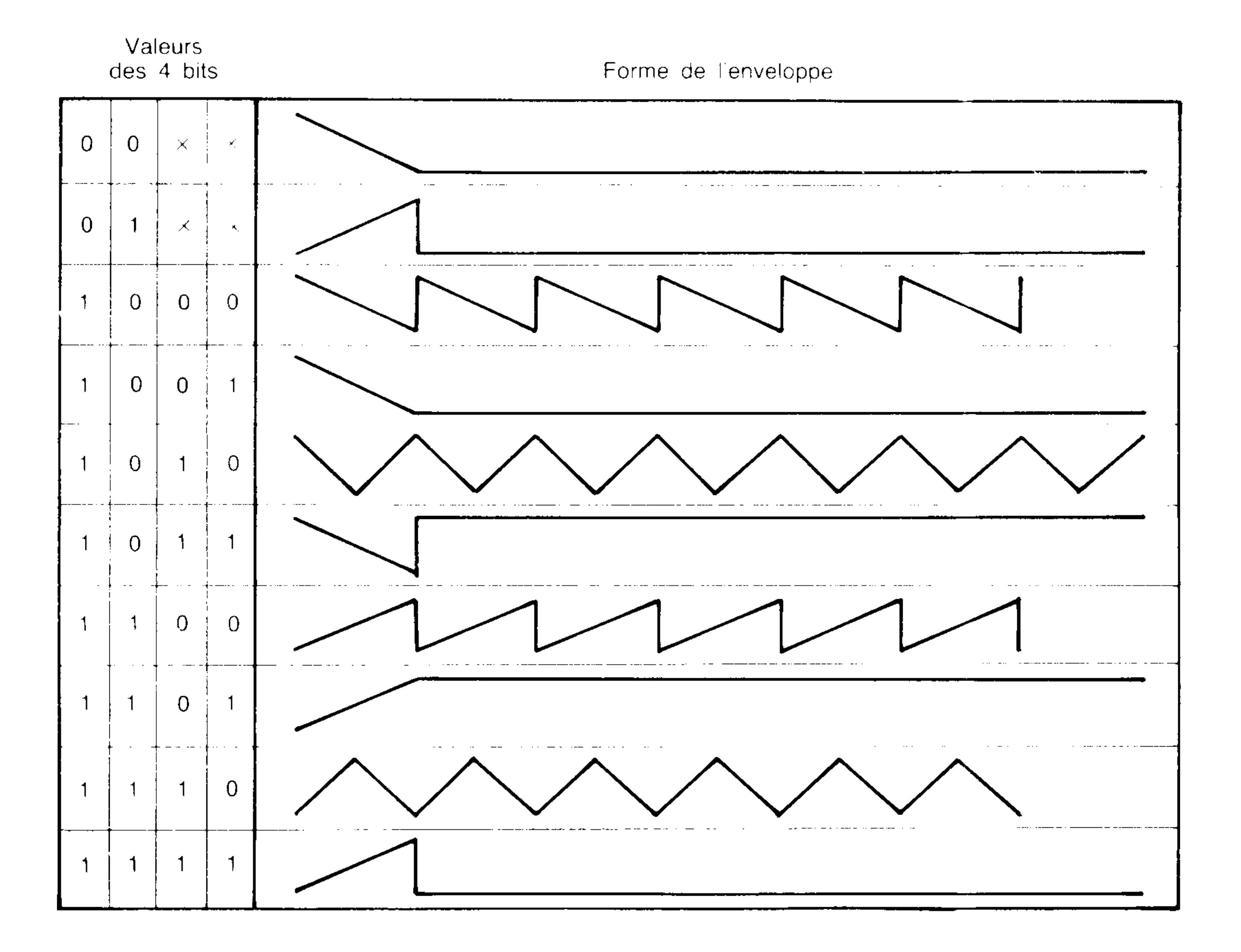
$$\text{Valeur} = \frac{3579545 \times \text{Période}}{256}$$

Les 8 bits de chacun des deux registres R11 et R12 sont utilisés. Ainsi, une valeur correspondant à une période donnée appartient à l'intervalle [1, 65535].

■ *Registre R13*

Ce registre sert à contrôler la forme de la modulation utilisée. Seuls les 4 bits les moins significatifs sont utilisés. Rappelez-vous que la modulation a lieu pour un canal si le bit 4 du registre d'amplitude correspondant est à 1. Voici un tableau vous permettant de trouver la forme de l'amplitude pour une valeur donnée des 4 premiers bits de ce registre.

Enveloppes



Les registres R14 et R15 seront vus un peu plus loin car ils servent en entrée/sortie pour le contrôle des joysticks.

Voyons pour l'instant comment produire des sons en ASSEMBLEUR.

En BASIC, la programmation du PSG peut se faire soit par l'instruction SOUND, soit par l'instruction OUT (En fait l'instruction SOUND est un ensemble d'instructions OUT.) En ASSEMBLEUR, nous allons nous servir de l'instruction OUT. Pour mettre une valeur dans un registre du PSG, il faut deux renseignements qui sont le numéro du registre et la donnée que l'on veut y mettre. La mise d'une donnée dans un registre comportera donc ces deux parties. La transmission de données au PSG se fait par l'intermédiaire des ports d'entrées sorties ayant pour adresses A0H et A1H. Sur le port d'adresse A0H, on doit transmettre le numéro du registre (compris entre 0 et 15) et sur le port d'adresse A1H on doit transmettre la donnée à mettre dans ce registre. On peut construire soi-même sa routine mais ceci n'est pas utile car nous vous donnons celle qui se trouve en ROM ; le vecteur du BIOS qui appelle cette dernière se trouve à l'adresse **0093H**. Il suffit de mettre le numéro du registre dans l'accumulateur et la donnée dans le registre E et cette routine se charge de faire le nécessaire. Nous vous donnons ici le désassemblage de cette routine pour que vous voyez comment elle fonctionne :

```
1102H    DI            ; Interdiction des interruptions
1103H    OUT (0A0H),A  ; Transmet le numéro du registre
1105H    PUSH AF       ; Conserver A
1106H    LD A,E        ; A := E
1107H    OUT (0A1H),A  ; Transmet la donnée
1109H    EI            ; Autorise les interruptions
110AH    POP AF        ; Restitue A
110BH    RET           ; Retour
```

Rien de plus simple pour vous maintenant que de jouer un air de musique en ASSEMBLEUR !!!

■ *Registres R14 et R15*

Le PSG possède deux registres d'entrées sorties qui servent dans les MSX à la gestion des joysticks et des manettes analogiques. Ces ports peuvent être dirigés en entrée ou en sortie comme nous vous le disions lors de la description du registre R7. En conséquence, on peut écrire ou lire sur ces ports mais seulement par l'intermédiaire de OUT et IN.

Ecriture : Comme pour les registres précédents, il faut fournir le numéro du registre et la donnée à écrire. La routine qui s'en charge à l'adresse 0093H est toujours utilisable.

Lecture : La démarche est un peu différente puisque les ports d'entrées/sorties utilisés ne sont plus A0H et A1H mais A0H et A2H. La lecture se fait en chargeant le numéro du registre à lire sur le port A0H (par exemple par OUT 0A0H,numéro) et l'on récupère le contenu du registre demandé sur le port A2H (par exemple par IN A,(0A2H)). Comme

pour l'écriture, il existe une routine qui fait cela toute seule en ROM. Celle-ci est appelée par un vecteur du BIOS qui se trouve à l'adresse **0096H**. Le paramètre d'entrée est le numéro du registre à lire qui doit se trouver dans l'accumulateur, et le paramètre de retour est le contenu de ce registre qui se trouve dans l'accumulateur. Ainsi, si vous voulez lire le contenu du port A vous faites :

```
LD A,14
CALL 0096H
```

Au retour le contenu du port A se trouvera dans l'accumulateur.

Que représentent ces ports ?
Voici un détail de toutes les fonctions de leurs bits :

Le port A (registre 14) sert principalement à la lecture des joysticks. Les bits b0, b1, b2 et b3 servent à déterminer la direction dans laquelle est poussée le manche à balai selon la convention qu'un bit passe à 0 lorsque l'on pousse le manche dans la direction qui lui correspond et avec :

Bit 0 correspondant au nord
Bit 1 correspondant au sud
Bit 2 correspondant à l'ouest
Bit 3 correspondant à l'est

Nota : Si vous poussez la manette dans la direction sud-est,les bits correspondant au sud et à l'est sont cumulés.
Le bit 4 sert pour le bouton de tir ; il est à 0 si le bouton est enfoncé.
Le bit 5 sert pour le trigger des manettes analogiques.
Le bit 6 ne sert pas.
Le bit 7 sert pour la cassette.

Le port B sert pour la sélection du joystick (comme on dispose de deux joysticks, il faut dire celui que l'on désire scruter) et pour les manettes analogiques. Voici le détail de ses bits : les bits b0 à b5 servent pour la valeur de la manette analogique. Cette valeur est donc comprise entre 0 et 31.
Le bit 6 sert à sélectionner le joystick que l'on veut lire (1 ou 2).
Le bit 7 n'est pas utilisé.

Pour scruter les joysticks, il existe en ROM des routines dont vous pouvez vous resservir. Ces routines sont :

Lecture du joystick : On met dans l'accumulateur le numéro (1 ou 2) de la manette que l'on désire lire ; on fait un CALL à l'adresse D5H. On récupère alors la valeur dans l'accumulateur.

Lecture du bouton de tir : On met dans A le numéro du joystick et l'on fait un CALL à l'adresse D8H.

Lecture de la manette analogique : La routine située en DBH est équivalente à la fonction PAD et celle située en DEH est équivalente à la fonction PDL.

Voici un tableau récapitulatif de tous les registres du PSG et de leurs fonctions respectives :

Fonctions des registres du P.S.G.

Registre \ bit	7	6	5	4	3	2	1	0
R0	Fréquence canal A bits 0 à 7							
R1					Fréquence canal A bits 8 à 11			
R2	Fréquence canal B bits 0 à 7							
R3					Fréquence canal B bits 8 à 11			
R4	Fréquence canal C bits 0 à 7							
R5					Fréquence canal C bits 8 à 11			
R6				Fréquence du bruit				
R7	Direction ports E S		Présence bruit sur canaux			Présence son sur canaux		
	B	A	C	B	A	C	B	A
R8				Modulation A	Amplitude canal A			
R9				Modulation B	Amplitude canal B			
R10				Modulation C	Amplitude canal C			
R11	Codage période de l'enveloppe bits 0 à 7							
R12	Codage période de l'enveloppe bits 8 à 15							
R13					Codage de la forme de l'enveloppe			
R14	E S port A							
R15	E S port B							

12

BIOS, ROM, et région de communication

Toutes les adresses seront données en hexadécimal. Nous oublierons le « H » qui l'indique (mais il est sous-entendu).

Dans le cas où nous indiquons les modifications des registres, seuls les registres indiqués sont modifiés, les autres sont préservés ; par contre, s'il n'y a pas d'indication, on ignore quels registres sont modifiés.

LE BIOS

Le BIOS est la région de la ROM qui est garantie par le standard MSX ; vous pouvez être certain que dans l'avenir, les adresses des routines qu'il contient ne changeront pas, ce qui n'est pas le cas de la ROM. Dans un programme destiné au commerce, il ne faut donc utiliser que le BIOS ; cependant dans le but de vous aider si vous désirez acquérir une connaissance approfondie du MSX, nous vous donnons les adresses où le BIOS renvoit dans la ROM actuelle (février 1985).

adresse	saut	fonction
01	02D7	Réinitialisation générale.
08	2683	RST8 : Vérification de syntaxe.
0C	1B6	Gestion des slots.
10	2986	RST 10 : Utilisé par le BASIC.
14	1D1	Sélection des slots.
18	1B45	RST 18 : Sortie sur le périphérique courant.
1C	217	Gestion des slots.

adresse	saut	fonction
20	146A	RST 20 : Comparaison de HL et DE : si HL=DE, Z=1 ; si HL<DE, Carry=1.
24	25E	Gestion des slots.
28	2689	RST 28 : Test de l'indicateur de type de donnée (F663) de DAC et positionne les indicateurs : S=entier, Z=chaîne, P=simple précision, NC=double précision.
30	205	RST 30 : Gestion des slots.
38	C3C	RST 38 : Gestion des interruptions en provenance du VDP.
3B	49D	PSG : Initialisation.
3E	139D	Initialisation des touches de fonctions
41	577	VDP : Interdiction de l'affichage.
44	570	VDP : Autorisation de l'affichage.
47	57F	VDP : Ecriture dans le registre N C du contenu de B ; modifie AF et B ; met à jour la RC.
4A	7D7	VDP : Lit la Vidéoram à l'adresse (HL) et retourne la valeur dans A. Modifie AF.
4D	7CD	VDP : Ecriture dans la VRAM à l'adresse (HL) de la donnée (A). Modifie AF.
50	7EC	VDP : Positionne l'adresse (HL) en lecture. Modifie AF.
53	7DF	VDP : Positionne l'adresse (HL) en écriture. Modifie AF.
56	815	VDP : Ecriture en VRAM de BC fois la valeur (A) à partir de l'adresse (HL). Modifie AF et BC.
59	70F	VDP : Transfert d'un bloc de la VRAM en RAM : adresse VRAM dans HL, destination RAM dans DE, longueur dans BC. Modifie tous les registres.
5C	744	VDP : Transfert d'un bloc de RAM en VRAM : adresse RAM dans HL, destination VRAM dans DE, longueur dans BC. Modifie tous les registres.
5F	84F	VDP : Initialisation du mode d'affichage contenu dans A (0 à 3). Modifie tous les registres.
62	7F7	VDP : Positionne les couleurs d'encre de fond et de bord en fonction des octets F3E9, F3EA et F3EB. Modifie tous les registres.
66	1398	Gestion des interruptions NMI pour les machines qui en sont pourvues (crochet en FDD6).
69	6A8	VDP : Initialisation des sprites. Modifie tous les registres.
6C	50E	VDP : Initialisation en mode TEXT. Modifie tous les registres.
6F	538	VDP : Initialisation en mode GR1. Modifie tous les registres.
72	5D2	VDP : Initialisation en mode GR2. Modifie tous les registres.
75	61F	VDP : Initialisation en mode MULT. Modifie tous les registres.
78	594	VDP : Force le mode TEXT. Modifie tous les registres.
7B	5B4	VDP : Force le mode GR1. Modifie tous les registres.
7E	602	VDP : Force le mode GR2. Modifie tous les registres.
81	659	VDP : Force le mode MULT. Modifie tous les registres.
84	6E4	VDP : Retourne dans HL l'adresse dans la TGS du sprite dont le numéro est dans A. Modifie AF, DE, HL.

adresse	saut	fonction
87	6F9	VDP : Idem 84 pour la TAS.
8A	704	VDP : Retourne dans A la taille des sprites (8 ou 32) et dans Carry le bit de taille. Modifie AF.
8D	1510	VDP : Ecriture d'un caractère en GR2. Code ASCII dans A, coordonnées comptées en cellules en FCB7 (X) et FCB9 (Y), couleur en F3E9. Ne modifie aucun registre.
90	4BD	PSG : Initialisation. Modifie tous les registres.
93	1102	PSG : Ecriture dans le registre numéro A de la valeur (E). Ne modifie aucun registre.
96	110E	PSG : Lecture du registre de numéro A. Retourne la valeur dans A. Modifie A.
99	11C4	PSG : Utilisé par le BASIC.
9C	D6A	KBD : Lit le clavier sans attente et retourne le code ASCII de la touche appuyée dans A ou met Z à 1 s'il n'y a pas de touche appuyée.
9F	10CB	KBD : Attend l'appui d'une touche et retourne le code ASCII dans A. Modifie AF.
A2	8BC	VDP : Affichage en TEXT d'un caractère dont le code ASCII est dans A à la position courante du curseur (F3DC et F3DD).
A5	85D	IMP : Ecrit le caractère dont le code est dans A sur l'imprimante. Modifie F.
A8	884	IMP : Test de l'imprimante : Si elle est prête, on a en retour 255 dans A et Z=0, sinon A=0 et Z=1. Modifie AF.
AB	89D	IMP : Conversion d'un caractère non imprimable.
AE	23BF	Entrée de texte : Met dans un buffer dont le sommet 1 est dans HL, un texte frappé au clavier et terminé par RETURN. La touche STOP met Carry à 1. Modifie tous les registres.
B1	23D5	Idem utilisé par le BASIC si F6AA est à 1 (pas mode AUTO).
B4	23CC	Entrée utilisée par le BASIC.
B7	46F	Teste l'appui de CTRL-STOP même si les interruptions sont interdites. Carry=1 si CTRL-STOP pressé. Modifie AF.
BA	3FB	Utilisé par le BASIC.
BD	10F9	. . . .
C0	1113	Beep. Modifie tous les registres.
C3	848	CLS. Modifie AF, BC, DE.
C6	88E	Positionne le curseur à la ligne (L), et à la colonne (H). Modifie AF.
C9	B26	Affiche les touches de fonction si F3DE n'est pas nul. Modifie tous les registres.
CC	B15	Efface l'affichage des touches de fonction. Modifie tous les registres.
CF	B2B	Affiche les touches de fonction. Modifie tous les registres.
D2	83B	Retour à l'ancien mode d'écran après un passage en GR2 ou en MULT (FCB0). Modifie tous les registres.
D5	11EE	Lecture des manettes de jeux (A = N° manette → A = valeur). Modifie tous les registres.
D8	1253	Lecture des boutons de tir des joysticks (A = N° manette donne A = 0 si pas pressé, FF sinon). Modifie AF.

adresse	saut	fonction
DB	12AC	Equivalent de la fonction BASIC PAD. A = N PAD → A = valeur lue. Modifie tous les registres.
DE	1273	Equivalent de la fonction BASIC PDL. A = N PDL → A = valeur lue. Modifie tous les registres.
E1	1A63	CASS : Démarre le moteur du magnétophone, et lit l'en-tête. Carry = 1 si erreur. Modifie tous les registres.
E4	1ABC	CASS: Lit 1 octet depuis la cassette, et le place dans A. Carry = 1 si erreur. Modifie tous les registres.
E7	19E9	CASS : Arrête le moteur du magnétophone. Ne modifie aucun registre.
EA	19F1	CASS : Démarre le moteur, et écrit l'en-tête. A = 0 donne un en-tête court, A = 0 donne un en-tête long. Carry = 1 si erreur. Modifie tous les registres.
ED	1A19	CASS : Ecriture de l'octet se trouvant dans A. Carry = 1 si erreur. Modifie tous les registres.
F0	19DD	CASS : Fin de l'écriture. Ne modifie aucun registre.
F3	1384	CASS : Arrêt (A = 0), démarrage (A = 1) ou secousse (A = 255) du moteur. Modifie AF.
F6	14EB	SON : Utilisé par le BASIC.
F9	1492	SON : Utilisé par le BASIC.
FC	16C5	Déplace l'accumulateur graphique d'un point à droite.
FF	16EE	Idem FC à gauche.
102	175D	Idem FC en haut.
105	173C	Idem 102 avec C=1 si un bord est atteint.
108	172A	Idem FC en bas.
10B	170A	Idem 108 avec C=1 si un bord est atteint.
10E	1599	BC:=BC mod 8 et DE:=DE mod 8.
111	15DF	Calcul de l'adressage en GR2 du point x,y (cf. partie VDP).
114	1639	Lecture de l'accumulateur graphique. Donne A=masque et HL=adresse. Modifie A et HL.
117	1640	Ecriture de l'accumulateur graphique. Ne modifie aucun registre.
11A	1676	Ecriture dans l'écran de l'accumulateur graphique en mode GR2 ou MULT.
11D	1647	Utilisé par le BASIC pour gérer les couleurs en haute résolution.
120	167E	Ecriture dans l'écran GR2 ou MULT de l'accumulateur graphique. Utilise l'octet F3F2.
123	1809	Remplissage (BASIC).
126	18C7	Utilisé par le BASIC.
129	18CF	Initialisation de la couleur de bord pour PAINT.
12C	18E4	Scrutation des points vers la droite.
12F	197A	Scrutation des points vers la gauche.
132	F3D	Si A=0 éteint le témoin CAPS, sinon l'allume. Modifie AF.
135	F7A	Si A=0 interdit le son, sinon l'autorise. Modifie AF.
138	144C	PPI : Lecture du port A. Résultat dans l'Accumulateur. Modifie A.
13B	144F	PPI : Ecriture dans le port A du contenu de l'Accumulateur. Ne modifie aucun registre.

adresse	saut	fonction
13E	1449	VDP : retourne dans A la valeur du registre d'état. Modifie A.
141	1452	PPI : Ecriture sur le port C de l'accumulateur (ligne à scruter) et retourne dans A la valeur lue sur port B (touches enfoncées). Modifie AF.
144	148A	Vecteur crochet pour extensions.
147	148E	Vecteur crochet pour extensions.
14A	145F	Test d'E/S de fichier. Modifie AF.
14D	1B63	Comme A8 avec caractère TAB converti en Spaces. Modifie F.
150	1470	Utilisé par le BASIC pour le son.
153	1474	Idem.
156	468	Efface le tampon clavier. Modifie HL.
159	1FF	Gestion des slots (extensions).

LA RÉGION DE COMMUNICATION

La région de communication contient les variables système. Voici les principales. Les adresses sont toutes en hexadécimal. Le deuxième chiffre est le nombre d'octets utilisés par chacune.

Toutes les adresses contenues dans cette région sont bien sur partie faible d'abord.

F380	11	Routine de lecture des slots (BANKO).
F385	7	Routine d'écriture des slots (BANKO).
F38C	14	Routine d'appel pour CALL.
F39A	20	Table des adresses USR définies par DEF USR = ; 2 octets par adresse, dans l'ordre 0 10.
F3AE	1	Longueur d'1 ligne en texte (39 en standard).
F3AF	1	Idem en Graphique 1 (31 en standard).
F3B0	1	Longueur de la ligne courante (WIDTH).
F3B1	1	Nombre de lignes affichées sur l'écran. Peut changer la taille de l'affichage.
F3B2	1	Nombre de caractères pour TAB (14 en standard).
F3B3	40	Adresses de toutes les tables dans tous les modes (cf. VDP).
F3DB	1	Bascule de Click des touches : 0 = off, 1 = on.
F3DC	1	Position verticale du curseur.
F3DD	1	Position horizontale du curseur.
F3DE	1	Bascule d'affichage des touches de fonction : 1 = visibles, 0 = pas visibles.
F3DF	8	Contenu des 8 registres du VDP dans l'ordre de 0 à 7 (cf. VDP).
F3E9	1	Couleur du texte (utilisé entre autres par 8DH).
F3EA	1	Couleur de fond.
F3EB	1	Couleur de bord.
F3F2	1	Octet de couleur utilisé par de nombreuses routines (Acc. Graphique).
F3F3	2	Adresses des tables pour QUEUTL.

F3F6	1	Intervalle de scrutation clavier.
F3F8	2	Adresse de l'octet à écrire dans le tampon clavier.
F3FA	2	Adresse de l'octet à lire dans le tampon clavier.
F3FC	20	Paramètres d'écriture et lecture Cassette.
F40F	5	Utilisé par RESUME NEXT.
FIN DES PARAMETRES INITIALISES A L'ALLUMAGE		
F414	1	Numéro de la dernière erreur.
F415	1	Position de la tête de l'imprimante.
F416	1	Sortie sur écran (0) ou sur imprimante (1). Utilisé par A5H.
F417	1	0 = imprimante MSX, 1 = imprimante non-MSX.
F418	1	Différent de 0 si le caractère à imprimer n'est pas ASCII.
F419	4	Utilisés par VAL.
F41C	2	Adresse de la ligne en cours d'exécution (BASIC).
F41F	316	Tampon du CRUNCHER (codage BASIC).
F55E	258	Tampon Clavier.
F662	1	Flag pour DIM.
F663	1	Flag qui indique le type de variable traité.
F666	2	Adresse du caractère courant du texte BASIC.
F672	2	Adresse la plus haute autorisée au BASIC (modifiée par CLEAR).
F674	2	Adresse maximale pour la pile (50 octets autorisés en standard). Modifié par CLEAR...
F676	2	Adresse fin BASIC.
F6A3	2	Adresse de la dernière ligne DATA lue.
F6AA	1	Flag 0 = AUTO, 1 = Pas AUTO.
F6AB	2	Numéro ligne courante pour AUTO.
F6AD	2	Valeur de l'incrément pour AUTO.
F6AF	2	Utilisé par RESUME.
F6B3	2	Numéro de la ligne de la dernière erreur.
F6B5	2	Numéro de la ligne courante (pour LIST et EDIT).
F6B9	2	Numéro ligne ou aller si erreur.
F6C0	2	Pointe la prochaine instruction.
F6C2	2	Adresse de la table des variables (change avec la taille du programme BASIC).
F6C4	2	Adresse de la table des variables tableaux (change avec la taille du programme BASIC).
F6C6	2	Adresse du début de l'espace disponible. Change à chaque fois qu'une nouvelle variable est mise dans les tables.
F6C8	2	Pointeur DATA.
F6CA	26	Attribue un type de variable à chaque lettre de l'alphabet (8 = double precision en standard).
F7C4	1	1 = TRON, 0 = TROFF.
F7C5	16	Utilisés par les calculs mathématiques.
F7F6	8	Accumulateur mathématique (DAC).
F847	8	Accumulateur mathématique secondaire (ARG).
F857	8	Contient le dernier nombre aléatoire généré.
F866	1	Différent de 0 si démarrage automatique après chargement.

F87F	160	Contenu des touches de fonction (F1-FI0).
F91F	2	Adresse du jeu de caractères en ROM.
F922	8	Adresses de tables courantes (cf. VDP).
F92A	2	Accumulateur Graphique : Adresse en VRAM (cf. VDP).
F92C	1	Accumulateur Graphique : Masque (cf. VDP).
F92D	41	Variables pour instructions graphiques (CIRCLE, PAINT).
F956		Variables pour les instructions de son.
FBB0	1	Pas 0 si démarrage à chaud autorisé.
FBB1	1	Pas 0 si le BASIC est en ROM.
FBCC	1	Code pour le curseur.
FBCD	1	Indique quelles touches de fonctions sont affichées.
FC48	2	Adresse début RAM installée.
FC4A	2	Adresse du haut de la RAM.
FC9C	1	Valeur Y de PAD.
FC9D	1	Valeur X de PAD.
FCA6	1	Flag de sortie un caractère graphique.
FCA8	1	Flag de mode insertion.
FCA9	1	Flag curseur ON/OFF.
FCAB	1	Flag CAPS.
FCAD	1	0 si charmant d'un programme BASIC.
FCAF	1	Mode courant de l'écran (0, 1, 2, ou 3). Très utilisé par les routines graphiques.
FCB0	1	Ancien mode de l'écran (auquel revenir après un passage en SCREEN 2 ou 3).
FCB2	1	Couleur de frontière pour PAINT.
FCB3	2	Position horizontale du curseur graphique.
FCB5	2	Position verticale du curseur graphique.
FCB7	2	Coordonnée X pour écriture en mode GR2.
FCB9	2	Coordonnée Y pour écriture en mode GR2.
FCBB	3	Paramètres pour DRAW.
FCBF	2	Adresse de lancement (BSAVE et BLOAD).
FCC1	4	Table des flags pour l'extension des slots.
FCC5	4	Etat courant de chaque registre d'extension de slot.
FCC9	64	Attributs pour chaque slot.
FD89	16	Nom de l'appel d'extension.
FD99	1	Identification de cartouche.

Vecteurs crochets de la région de communication

ad.	fonction
FD9A	Traitement des interruption en provenance du VDP.
FD9F	Idem FD9A.
FDA4	Appelé lors de l'écriture en mode texte du contenu de A.
FDA9	Appelé à l'affichage du curseur.
FDAE	Appelé à l'effacement du curseur.

ad.	fonction
FDB3	Appelé lors de l'affichage des touches de fonctions.
FDB8	Appelé lors de l'effacement de la signification des touches de fonctions.
FDBD	Appelé lorsque le mode texte est forcé.
FDC2	Appelé lors de la lecture d'un caractère.
FDC7	Appelé lors de l'initialisation VDP.
FDCC	Appelé lors de la lecture clavier.
FDD1	Appelé lors de la conversion du code matriciel du caractère.
FDD6	Appelé lors des interruptions NMI.
FDDB	Appelé lors de l'entrée d'une ligne.
FDE0	Appelé lors du ? des questions (type INPUT).
FDE5	Appelé lors d'une entrée ligne.
FDEA	Appelé lors de ON GOTO ou ON GOSUB.
FDEF	Appelé par DSKO$.
FDF4	Appelé par SETS.
FDF9	Appelé par NAME.
FDFE	Appelé par KILL.
FE03	Appelé par IPL.
FE08	Appelé par COPY.
FE0D	Appelé par CMD.
FE12	Appelé par DSKF.
FE17	Appelé par DSKI.
FE1C	Appelé par ATTR$.
FE21	Appelé par LSET.
FE26	Appelé par RSET.
FE2B	Appelé par FIELD.
FE30	Appelé par MKI$.
FE35	Appelé par MKS$.
FE3A	Appelé par MKD$.
FE3F	Appelé par CVI.
FE44	Appelé par CVS.
FE49	Appelé par CVD.
FE4E	Appelé par la routine GETPTR *(Get file pointer).*
FE53	Appelé par la routine SETFIL *(Set file pointer).*
FE58	Appelé par OPEN.
FE5D	Appelé par KILL, LOAD, MERGE.
FE62	Appelé par CLOSE.
FE67	Appelé par MERGE.
FE6C	Appelé par SAVE.
FE71	Appelé par BSAVE.
FE76	Appelé par BLOAD.
FE7B	Appelé par FILES.
FE80	Appelé par DGET *(Disk get).*
FE85	Appelé par FILOU *(File out).*
FE8A	Appelé par INDSKC *(Input disk character).*
FE8F	Appelé par INPUT$.

ad.	fonction
FE94	Appelé pour une sauvegarde.
FE99	Appelé par LOC.
FE9E	Appelé par LOF.
FEA3	Appelé par EOF.
FEA8	Appelé par FPOS *(File position).*
FEAD	Appelé par le BACK UP.
FEB2	Appelé par PARDEV *(Parse device name).* —
FEB7	Appelé par NODEVN *(No device name).* _
FEBC	Appelé par POSDSK *(Possibly disk).* —
FEC1	Appelé par DEVNAM *(Device name).*
FEC6	Appelé par GENDSP *(General device dispatcher).*
FECB	Appelé par RUNC *(Run clear).*
FED0	Appelé lors de l'effacement des variables.
FED5	Appelé lors de l'initialisation de la table des variables.
FEDA	Appelé lors d'une erreur de pile.
FEDF	Appelé lors du test de fichier.
FEE4	Appelé lors d'une sortie sur écran ou imprimante.
FEE9	Appelé lors d'un retour chariot suivi d'un saut de ligne.
FEEE	Appelé lors de la lecture d'une donnée sur disque.
FEF3	Appelé lors des fonctions graphiques.
FEF8	Appelé lors d'un END.
FEFD	Appelé lors d'une erreur (pour son impression).
FF02	Idem.
FF07	Appelé lors du READY.
FF0C	Appelé lors d'un interprétation.
FF11	Appelé lors du traitement en mode direct.
FF16	Appelé en fin d'interprétation.
FF1B	Idem.
FF20	Appelé lors de la Tokenisation.
FF25	Idem.
FF2A	Idem.
FF2F	Idem.
FF34	Idem.
FF39	Appelé pour les routines mathématiques.
FF3E	Appelé lors d'une nouvelle instruction.
FF43	Appelé lors de rupture de séquence.
FF48	Appelé lors d'une saisie de caractère.
FF4D	Appelé lors du RETURN.
FF52	Appelé lors du PRINT.
FF57	Idem.
FF5C	Idem.
FF61	Appelé par DATA.
FF66	Appelé lors de l'évaluation d'une formule.
FF6B	Appelé pour les calculs mathématiques.
FF70	Appelé lors de l'évaluation d'une expression.

ad.	fonction
FF75	Appelé lors des fonction mathématiques transcendantes.
FF7A	Idem.
FF7F	Appelé par MID$.
FF84	Appelé par WIDTH.
FF89	Appelé par LIST.
FF8E	Appelé lors de la détokenisation.
FF93	Appelé par POKE.
FF98	Appelé lors de la gestion des lignes (numéro ←→ pointeur).
FF9D	Appelé lors de recherche de place pour création de nouvelle chaine.
FFA2	Appelé lors de la lecture d'une variable.
FFA7	Appelé par le BIOS.
FFAC	Appelé par le BIOS.
FFB1	Appelé lors du traitement des erreurs.
FFB6	Appelé lors d'une impression sur imprimante.
FFBB	Appelé lors du test de l'imprimante.
FFC0	Appelé par SCREEN.
FFC5	Appelé par PLAY.
FFCA	Fin de la zone de travail.

Annexe 1

Corrections des exercices

Exercice 1.1

a) 170
b) AAH
c) 7
255

Exercice 1.2 :

Le système hexadécimal est une numérotation positionnelle à base 16. Le poids de chaque chiffre est (en partant de la droite) 1, 16, 16^2 soit 256, 16^3 soit 4096.

Exercice 1.3 :

a) Il y a 4 pages dans 1Ko. En effet, une page contient 256 octets et 1Ko en contient 1024, soit 4 fois plus.
b) Il y a 256 octets par page donc FFH en hexadécimal.
c) Dans 1Ko, il y a 1024 octets soit 400H en hexadécimal

Exercice 3.1:

a) Résultat : 10001001 en binaire
soit 89H en hexadécimal
soit 137 en décimal

b) Résultat : 10011001 en binaire
soit 99H en hexadécimal
soit 153 en décimal

Exercice 3.2 :

a) −4 + 5 :

```
      11111011
  +   00000101
(1)   00000000
```

soit 0 en décimal, ce qui est inexact.

b) 2 − 6 :

```
      00000010
  +   10000110
  =   10001000
```

soit −119 en décimal, ce qui est inexact.

Exercice 3.3 :

a) 5 − 2 :

```
      00000101
  +   11111110
(1)   00000011
```

soit 3 en décimal, ce qui est exact si l'on ne tient pas compte de la retenue.

b) 4 + 6 :

```
      00000100
  +   00000110
  =   00001010
```

soit 10 en décimal, ce qui est exact.

c) 3 − 7 :

```
      00000011
  +   11111001
  =   11111100
```

soit −4 en décimal, ce qui est exact.

d) −3 − 3 :

```
      11111101
  +   11111101
(1)   11111010
```

soit −6 en décimal, ce qui est exact.

Exercice 3.4 :

a) 54 est codé 01010100 en DCB
b) 97 est codé 10010111 en DCB
c) 12 est codé 00010010 en DCB
d) 05 est codé 00000101 en DCB
e) 114 est codé 000100010100 en DCB. Il ne tient pas sur un octet car il est supérieur à 99.

Exercice 3.5 :

```
      10111001
AND   00111101
  =   00111001
```

Exercice 3.6 :

```
      00010111
  OR  10101011
  =   10111111
```

Exercice 3.7 :

```
      10111001
 XOR  10000111
  =   00111110
```

Exercice 5.1 :

LD D,90H	Met la donnée immédiate 90H dans le registre D. Adressage immédiat.
LD E,0	Met la donnée immédiate 0 dans le registre E.
EX DE,HL	Echange les contenus des doubles registres DE et HL. Adressage registre. Ici HL=9000H.
LD (HL),03	Charge la case pointée par le double registre HL de la valeur immédiate 03. L'adressage est donc indirect et immédiat. Ici (9000H)=03.
LD IX,9005H	Charge le registre d'index IX de la valeur immédiate 9005H. Adressage immédiat.
LD B,(IX−5)	Charge le registre B par la donnée se trouvant à l'adresse IX−5 (IX contient un nombre de 16 bits). Adressage indirect. Ici B=(9000H) soit B=03.
LD A,07	Charge l'accumulateur avec la donnée immédiate 07. Adressage immédiat.
CP B	Compare le registre B à l'accumulateur. Adressage registre.
JR NZ,ICI	Saute au label ICI puisque l'indicateur Z n'est pas positionné du fait que B (qui vaut 03) est différent de A (qui vaut 07). Adressage relatif.
PUSH HL	Empile le double registre HL. Adressage indirect.
POP BC	Dépile le sommet de pile dans le double registre BC. Adressage indirect. Le but de ces deux dernières instructions a été de transférer le contenu de HL dans BC.

Annexe 2

Codes ASCII

NUL : Null
SOH : Start of Heading
STX : Start of Text
ETX : End of Text
EOT : End of Tansmission
ENQ : Enquiry
ACK : Acknowledge
BEL : Bell
BS : Backspace
HT : Horizontal Tabulation
LF : Line Feed
VT : Vertical Tabulation
FF : Form Feed
CR : Carriage Return
SO : Shift Out
SI : Shift in
DLE : Data Link Escape
DC : Device Control
NAK : Negative Acknowledge
SYN : Synchronous idle
ETB : End of Transmission Block
CAN : Cancel
EM : End of Medium
SUB : Substitute
ESC : Escape
FS : File Separator
GS : Group Separator
RS : Record Separator
US : Unit Separator
SPC : Space

MSB \ LSB	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
#	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SPC	!	"	#	$	%	&	`	(	)	*	+	,	—	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	—
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	:	}	~	◆

Annexe 3

Tableau de conversion

Valeur de l'octet				Complément à un			Complément à deux		
Dec.	ASC	Hexa.	Binaire	Dec.	Hexa.	Binaire	Dec.	Hexa.	Binaire
0		0	00000000	255	FF	11111111	0	0	00000000
1		1	00000001	254	FE	11111110	255	FF	11111111
2		2	00000010	253	FD	11111101	254	FE	11111110
3		3	00000011	252	FC	11111100	253	FD	11111101
4		4	00000100	251	FB	11111011	252	FC	11111100
5		5	00000101	250	FA	11111010	251	FB	11111011
6		6	00000110	249	F9	11111001	250	FA	11111010
7		7	00000111	248	F8	11111000	249	F9	11111001
8		8	00001000	247	F7	11110111	248	F8	11111000
9		9	00001001	246	F6	11110110	247	F7	11110111
10		A	00001010	245	F5	11110101	246	F6	11110110
11		B	00001011	244	F4	11110100	245	F5	11110101
12		C	00001100	243	F3	11110011	244	F4	11110100
13		D	00001101	242	F2	11110010	243	F3	11110011
14		E	00001110	241	F1	11110001	242	F2	11110010
15		F	00001111	240	F0	11110000	241	F1	11110001
16		10	00010000	239	EF	11101111	240	F0	11110000
17		11	00010001	238	EE	11101110	239	EF	11101111
18		12	00010010	237	ED	11101101	238	EE	11101110
19		13	00010011	236	EC	11101100	237	ED	11101101
20		14	00010100	235	EB	11101011	236	EC	11101100
21		15	00010101	234	EA	11101010	235	EB	11101011
22		16	00010110	233	E9	11101001	234	EA	11101010
23		17	00010111	232	E8	11101000	233	E9	11101001
24		18	00011000	231	E7	11100111	232	E8	11101000
25		19	00011001	230	E6	11100110	231	E7	11100111
26		1A	00011010	229	E5	11100101	230	E6	11100110
27		1B	00011011	228	E4	11100100	229	E5	11100101
28		1C	00011100	227	E3	11100011	228	E4	11100100

Valeur de l'octet				Complément à un			Complément à deux		
Dec.	ASC	Hexa.	Binaire	Dec.	Hexa.	Binaire	Dec.	Hexa.	Binaire
29		1D	00011101	226	E2	11100010	227	E3	11100011
30		1E	00011110	225	E1	11100001	226	E2	11100010
31		1F	00011111	224	E0	11100000	225	E1	11100001
32		20	00100000	223	DF	11011111	224	E0	11100000
33	!	21	00100001	222	DE	11011110	223	DF	11011111
34	"	22	00100010	221	DD	11011101	222	DE	11011110
35	#	23	00100011	220	DC	11011100	221	DD	11011101
36	$	24	00100100	219	DB	11011011	220	DC	11011100
37	%	25	00100101	218	DA	11011010	219	DB	11011011
38	&	26	00100110	217	D9	11011001	218	DA	11011010
39	'	27	00100111	216	D8	11011000	217	D9	11011001
40	(	28	00101000	215	D7	11010111	216	D8	11011000
41	)	29	00101001	214	D6	11010110	215	D7	11010111
42	*	2A	00101010	213	D5	11010101	214	D6	11010110
43	+	2B	00101011	212	D4	11010100	213	D5	11010101
44	,	2C	00101100	211	D3	11010011	212	D4	11010100
45	-	2D	00101101	210	D2	11010010	211	D3	11010011
46	.	2E	00101110	209	D1	11010001	210	D2	11010010
47	/	2F	00101111	208	D0	11010000	209	D1	11010001
48	0	30	00110000	207	CF	11001111	208	D0	11010000
49	1	31	00110001	206	CE	11001110	207	CF	11001111
50	2	32	00110010	205	CD	11001101	206	CE	11001110
51	3	33	00110011	204	CC	11001100	205	CD	11001101
52	4	34	00110100	203	CB	11001011	204	CC	11001100
53	5	35	00110101	202	CA	11001010	203	CB	11001011
54	6	36	00110110	201	C9	11001001	202	CA	11001010
55	7	37	00110111	200	C8	11001000	201	C9	11001001
56	8	38	00111000	199	C7	11000111	200	C8	11001000
57	9	39	00111001	198	C6	11000110	199	C7	11000111
58	:	3A	00111010	197	C5	11000101	198	C6	11000110
59	;	3B	00111011	196	C4	11000100	197	C5	11000101
60	<	3C	00111100	195	C3	11000011	196	C4	11000100
61	=	3D	00111101	194	C2	11000010	195	C3	11000011
62	>	3E	00111110	193	C1	11000001	194	C2	11000010
63	?	3F	00111111	192	C0	11000000	193	C1	11000001
64	@	40	01000000	191	BF	10111111	192	C0	11000000
65	A	41	01000001	190	BE	10111110	191	BF	10111111
66	B	42	01000010	189	BD	10111101	190	BE	10111110
67	C	43	01000011	188	BC	10111100	189	BD	10111101
68	D	44	01000100	187	BB	10111011	188	BC	10111100
69	E	45	01000101	186	BA	10111010	187	BB	10111011
70	F	46	01000110	185	B9	10111001	186	BA	10111010
71	G	47	01000111	184	B8	10111000	185	B9	10111001
72	H	48	01001000	183	B7	10110111	184	B8	10111000
73	I	49	01001001	182	B6	10110110	183	B7	10110111
74	J	4A	01001010	181	B5	10110101	182	B6	10110110
75	K	4B	01001011	180	B4	10110100	181	B5	10110101
76	L	4C	01001100	179	B3	10110011	180	B4	10110100
77	M	4D	01001101	178	B2	10110010	179	B3	10110011
78	N	4E	01001110	177	B1	10110001	178	B2	10110010
79	O	4F	01001111	176	B0	10110000	177	B1	10110001
80	P	50	01010000	175	AF	10101111	176	B0	10110000
81	Q	51	01010001	174	AE	10101110	175	AF	10101111
82	R	52	01010010	173	AD	10101101	174	AE	10101110
83	S	53	01010011	172	AC	10101100	173	AD	10101101
84	T	54	01010100	171	AB	10101011	172	AC	10101100
85	U	55	01010101	170	AA	10101010	171	AB	10101011

Valeur de l'octet				Complément à un			Complément à deux		
Dec.	ASC	Hexa.	Binaire	Dec.	Hexa.	Binaire	Dec.	Hexa.	Binaire
86	V	56	01010110	169	A9	10101001	170	AA	10101010
87	W	57	01010111	168	A8	10101000	169	A9	10101001
88	X	58	01011000	167	A7	10100111	168	A8	10101000
89	Y	59	01011001	166	A6	10100110	167	A7	10100111
90	Z	5A	01011010	165	A5	10100101	166	A6	10100110
91	[	5B	01011011	164	A4	10100100	165	A5	10100101
92	\	5C	01011100	163	A3	10100011	164	A4	10100100
93	]	5D	01011101	162	A2	10100010	163	A3	10100011
94	^	5E	01011110	161	A1	10100001	162	A2	10100010
95	_	5F	01011111	160	A0	10100000	161	A1	10100001
96	`	60	01100000	159	9F	10011111	160	A0	10100000
97	a	61	01100001	158	9E	10011110	159	9F	10011111
98	b	62	01100010	157	9D	10011101	158	9E	10011110
99	c	63	01100011	156	9C	10011100	157	9D	10011101
100	d	64	01100100	155	9B	10011011	156	9C	10011100
101	e	65	01100101	154	9A	10011010	155	9B	10011011
102	f	66	01100110	153	99	10011001	154	9A	10011010
103	g	67	01100111	152	98	10011000	153	99	10011001
104	h	68	01101000	151	97	10010111	152	98	10011000
105	i	69	01101001	150	96	10010110	151	97	10010111
106	j	6A	01101010	149	95	10010101	150	96	10010110
107	k	6B	01101011	148	94	10010100	149	95	10010101
108	l	6C	01101100	147	93	10010011	148	94	10010100
109	m	6D	01101101	146	92	10010010	147	93	10010011
110	n	6E	01101110	145	91	10010001	146	92	10010010
111	o	6F	01101111	144	90	10010000	145	91	10010001
112	p	70	01110000	143	8F	10001111	144	90	10010000
113	q	71	01110001	142	8E	10001110	143	8F	10001111
114	r	72	01110010	141	8D	10001101	142	8E	10001110
115	s	73	01110011	140	8C	10001100	141	8D	10001101
116	t	74	01110100	139	8B	10001011	140	8C	10001100
117	u	75	01110101	138	8A	10001010	139	8B	10001011
118	v	76	01110110	137	89	10001001	138	8A	10001010
119	w	77	01110111	136	88	10001000	137	89	10001001
120	x	78	01111000	135	87	10000111	136	88	10001000
121	y	79	01111001	134	86	10000110	135	87	10000111
122	z	7A	01111010	133	85	10000101	134	86	10000110
123	{	7B	01111011	132	84	10000100	133	85	10000101
124	|	7C	01111100	131	83	10000011	132	84	10000100
125	}	7D	01111101	130	82	10000010	131	83	10000011
126	~	7E	01111110	129	81	10000001	130	82	10000010
127	♦	7F	01111111	128	80	10000000	129	81	10000001
128		80	10000000	127	7F	01111111	128	80	10000000
129		81	10000001	126	7E	01111110	127	7F	01111111
130		82	10000010	125	7D	01111101	126	7E	01111110
131		83	10000011	124	7C	01111100	125	7D	01111101
132		84	10000100	123	7B	01111011	124	7C	01111100
133		85	10000101	122	7A	01111010	123	7B	01111011
134		86	10000110	121	79	01111001	122	7A	01111010
135		87	10000111	120	78	01111000	121	79	01111001
136		88	10001000	119	77	01110111	120	78	01111000
137		89	10001001	118	76	01110110	119	77	01110111
138		8A	10001010	117	75	01110101	118	76	01110110
139		8B	10001011	116	74	01110100	117	75	01110101
140		8C	10001100	115	73	01110011	116	74	01110100
141		8D	10001101	114	72	01110010	115	73	01110011
142		8E	10001110	113	71	01110001	114	72	01110010

Valeur de l'octet				Complément à un			Complément à deux		
Dec.	ASC	Hexa.	Binaire	Dec.	Hexa.	Binaire	Dec.	Hexa.	Binaire
143		8F	10001111	112	70	01110000	113	71	01110001
144		90	10010000	111	6F	01101111	112	70	01110000
145		91	10010001	110	6E	01101110	111	6F	01101111
146		92	10010010	109	6D	01101101	110	6E	01101110
147		93	10010011	108	6C	01101100	109	6D	01101101
148		94	10010100	107	6B	01101011	108	6C	01101100
149		95	10010101	106	6A	01101010	107	6B	01101011
150		96	10010110	105	69	01101001	106	6A	01101010
151		97	10010111	104	68	01101000	105	69	01101001
152		98	10011000	103	67	01100111	104	68	01101000
153		99	10011001	102	66	01100110	103	67	01100111
154		9A	10011010	101	65	01100101	102	66	01100110
155		9B	10011011	100	64	01100100	101	65	01100101
156		9C	10011100	99	63	01100011	100	64	01100100
157		9D	10011101	98	62	01100010	99	63	01100011
158		9E	10011110	97	61	01100001	98	62	01100010
159		9F	10011111	96	60	01100000	97	61	01100001
160		A0	10100000	95	5F	01011111	96	60	01100000
161		A1	10100001	94	5E	01011110	95	5F	01011111
162		A2	10100010	93	5D	01011101	94	5E	01011110
163		A3	10100011	92	5C	01011100	93	5D	01011101
164		A4	10100100	91	5B	01011011	92	5C	01011100
165		A5	10100101	90	5A	01011010	91	5B	01011011
166		A6	10100110	89	59	01011001	90	5A	01011010
167		A7	10100111	88	58	01011000	89	59	01011001
168		A8	10101000	87	57	01010111	88	58	01011000
169		A9	10101001	86	56	01010110	87	57	01010111
170		AA	10101010	85	55	01010101	86	56	01010110
171		AB	10101011	84	54	01010100	85	55	01010101
172		AC	10101100	83	53	01010011	84	54	01010100
173		AD	10101101	82	52	01010010	83	53	01010011
174		AE	10101110	81	51	01010001	82	52	01010010
175		AF	10101111	80	50	01010000	81	51	01010001
176		B0	10110000	79	4F	01001111	80	50	01010000
177		B1	10110001	78	4E	01001110	79	4F	01001111
178		B2	10110010	77	4D	01001101	78	4E	01001110
179		B3	10110011	76	4C	01001100	77	4D	01001101
180		B4	10110100	75	4B	01001011	76	4C	01001100
181		B5	10110101	74	4A	01001010	75	4B	01001011
182		B6	10110110	73	49	01001001	74	4A	01001010
183		B7	10110111	72	48	01001000	73	49	01001001
184		B8	10111000	71	47	01000111	72	48	01001000
185		B9	10111001	70	46	01000110	71	47	01000111
186		BA	10111010	69	45	01000101	70	46	01000110
187		BB	10111011	68	44	01000100	69	45	01000101
188		BC	10111100	67	43	01000011	68	44	01000100
189		BD	10111101	66	42	01000010	67	43	01000011
190		BE	10111110	65	41	01000001	66	42	01000010
191		BF	10111111	64	40	01000000	65	41	01000001
192		C0	11000000	63	3F	00111111	64	40	01000000
193		C1	11000001	62	3E	00111110	63	3F	00111111
194		C2	11000010	61	3D	00111101	62	3E	00111110
195		C3	11000011	60	3C	00111100	61	3D	00111101
196		C4	11000100	59	3B	00111011	60	3C	00111100
197		C5	11000101	58	3A	00111010	59	3B	00111011
198		C6	11000110	57	39	00111001	58	3A	00111010
199		C7	11000111	56	38	00111000	57	39	00111001

Valeur de l'octet				Complément à un			Complément à deux		
Dec.	ASC	Hexa.	Binaire	Dec.	Hexa.	Binaire	Dec.	Hexa.	Binaire
200		C8	11001000	55	37	00110111	56	38	00111000
201		C9	11001001	54	36	00110110	55	37	00110111
202		CA	11001010	53	35	00110101	54	36	00110110
203		CB	11001011	52	34	00110100	53	35	00110101
204		CC	11001100	51	33	00110011	52	34	00110100
205		CD	11001101	50	32	00110010	51	33	00110011
206		CE	11001110	49	31	00110001	50	32	00110010
207		CF	11001111	48	30	00110000	49	31	00110001
208		D0	11010000	47	2F	00101111	48	30	00110000
209		D1	11010001	46	2E	00101110	47	2F	00101111
210		D2	11010010	45	2D	00101101	46	2E	00101110
211		D3	11010011	44	2C	00101100	45	2D	00101101
212		D4	11010100	43	2B	00101011	44	2C	00101100
213		D5	11010101	42	2A	00101010	43	2B	00101011
214		D6	11010110	41	29	00101001	42	2A	00101010
215		D7	11010111	40	28	00101000	41	29	00101001
216		D8	11011000	39	27	00100111	40	28	00101000
217		D9	11011001	38	26	00100110	39	27	00100111
218		DA	11011010	37	25	00100101	38	26	00100110
219		DB	11011011	36	24	00100100	37	25	00100101
220		DC	11011100	35	23	00100011	36	24	00100100
221		DD	11011101	34	22	00100010	35	23	00100011
222		DE	11011110	33	21	00100001	34	22	00100010
223		DF	11011111	32	20	00100000	33	21	00100001
224		E0	11100000	31	1F	00011111	32	20	00100000
225		E1	11100001	30	1E	00011110	31	1F	00011111
226		E2	11100010	29	1D	00011101	30	1E	00011110
227		E3	11100011	28	1C	00011100	29	1D	00011101
228		E4	11100100	27	1B	00011011	28	1C	00011100
229		E5	11100101	26	1A	00011010	27	1B	00011011
230		E6	11100110	25	19	00011001	26	1A	00011010
231		E7	11100111	24	18	00011000	25	19	00011001
232		E8	11101000	23	17	00010111	24	18	00011000
233		E9	11101001	22	16	00010110	23	17	00010111
234		EA	11101010	21	15	00010101	22	16	00010110
235		EB	11101011	20	14	00010100	21	15	00010101
236		EC	11101100	19	13	00010011	20	14	00010100
237		ED	11101101	18	12	00010010	19	13	00010011
238		EE	11101110	17	11	00010001	18	12	00010010
239		EF	11101111	16	10	00010000	17	11	00010001
240		F0	11110000	15	F	00001111	16	10	00010000
241		F1	11110001	14	E	00001110	15	F	00001111
242		F2	11110010	13	D	00001101	14	E	00001110
243		F3	11110011	12	C	00001100	13	D	00001101
244		F4	11110100	11	B	00001011	12	C	00001100
245		F5	11110101	10	A	00001010	11	B	00001011
246		F6	11110110	9	9	00001001	10	A	00001010
247		F7	11110111	8	8	00001000	9	9	00001001
248		F8	11111000	7	7	00000111	8	8	00001000
249		F9	11111001	6	6	00000110	7	7	00000111
250		FA	11111010	5	5	00000101	6	6	00000110
251		FB	11111011	4	4	00000100	5	5	00000101
252		FC	11111100	3	3	00000011	4	4	00000100
253		FD	11111101	2	2	00000010	3	3	00000011
254		FE	11111110	1	1	00000001	2	2	00000010
255		FF	11111111	0	0	00000000	1	1	00000001

Annexe 4

Les ports d'entrées/sorties

Périph.	**Ad.**	**Fonction**
VDP	98H	Ecriture/Lecture dans un registre du VDP
VDP	99H	Ecriture/Lecture du registre de commande du VDP
PSG	A0H	Ecriture du numéro du registre du PSG désiré
PSG	A1H	Ecriture d'une donnée dans un registre du PSG
PSG	A2H	Lecture des ports du PSG
PPI	A8H	Ecriture/Lecture sur le port A du PPI
PPI	A9H	Ecriture/Lecture sur le port B du PPI
PPI	AAH	Ecriture/Lecture sur le port C du PPI
PPI	ABH	Ecriture/Lecture du registre de commande du PPI
RS232	80H	Ecriture/Lecture de données sur le 8251
RS232	81H	Lecture du registre de commande du 8251
RS232	82H	Lecture des interrupteurs vitesse
RS232	83H	Lecture des interrupteurs mode Ecriture du bit de masque d'interruption
RS232	84H	Ecriture/Lecture du timer 8253 (partie 1)
RS232	85H	Ecriture/Lecture du timer 8253 (partie 2)
RS232	86H	Ecriture/Lecture du timer 8253 (partie 3)
RS232	87H	Ecriture de commande du 8253
IMPR	90H	Lecture du signal BUSY de l'imprimante Ecriture du signal STROBE
IMPR	91H	Ecriture d'un caractère sur l'imprimante
L.PEN	B0H ⋮ BBH	Ecriture/Lecture du crayon optique

Annexe 5

Tableau d'assemblage

Codes des indicateurs d'état :

*	Drapeau affecté par l'opération
0	Drapeau mis à 0 par l'opération
1	Drapeau mis à 1 par l'opération
?	Drapeau affecté de manière non significative par l'opération
rien	Drapeau non affecté par l'opération

Abréviations utilisées pour les opérandes : (pour des raisons techniques, elles sont différentes de celles du chapitre 6.)

op	Valeur sur 8 bits
ad	Valeur sur 16 bits

Mnémonique	Code objet (hexa)	Cycles	Registre d'état S	Z	H	P/V	N	C	Adressage	Page
ADC A, (HL)	8E	7	*	*	*	*	0	*	indirect	59
ADC A, (IX+op)	DD8Eop	19	*	*	*	*	0	*	indexé	"
ADC A, (IY+op)	FD8Eop	19	*	*	*	*	0	*	"	"
ADC A,A	8F	4	*	*	*	*	0	*	registre	"
ADC A,B	88	4	*	*	*	*	0	*	"	"
ADC A,C	89	4	*	*	*	*	0	*	"	"
ADC A,D	8A	4	*	*	*	*	0	*	"	"
ADC A,E	8B	4	*	*	*	*	0	*	"	"
ADC A,H	8C	4	*	*	*	*	0	*	"	"
ADC A,L	8D	4	*	*	*	*	0	*	"	"
ADC A,op	CEop	7	*	*	*	*	0	*	immédiat	"
ADC HL,BC	ED4A	15	*	*	?	*	0	*	registre	60

Mnémonique	Code objet (hexa)	Cycles	S	Z	H	P/V	N	C	Adressage	Page
ADC HL,DE	ED5A	15	*	*	?	*	0	*	registre	60
ADC HL,HL	ED6A	15	*	*	?	*	0	*	"	"
ADC HL,SP	ED7A	15	*	*	?	*	0	*	"	"
ADD A, (HL)	86	7	*	*	*	*	0	*	indirect	61
ADD A, (IX+op)	DD86op	19	*	*	*	*	0	*	indexé	"
ADD A, (IY+op)	FD86op	19	*	*	*	*	0	*	"	"
ADD A,A	87	4	*	*	*	*	0	*	registre	"
ADD A,B	80	4	*	*	*	*	0	*	"	"
ADD A,C	81	4	*	*	*	*	0	*	"	"
ADD A,D	82	4	*	*	*	*	0	*	"	"
ADD A,E	83	4	*	*	*	*	0	*	"	"
ADD A,H	84	4	*	*	*	*	0	*	"	"
ADD A,L	85	4	*	*	*	*	0	*	"	"
ADD A,op	C6op	7	*	*	*	*	0	*	immédiat	"
ADD HL,BC	ED4A	11			?		0	*	registre	62
ADD HL,DE	ED5A	11			?		0	*	"	"
ADD HL,HL	ED6A	11			?		0	*	"	"
ADD HL,SP	ED7A	11			?		0	*	"	"
ADD IX,BC	DD09	15			?		0	*	"	"
ADD IX,DE	DD19	15			?		0	*	"	"
ADD IX,IX	DD29	15			?		0	*	"	"
ADD IX,SP	DD39	15			?		0	*	"	"
ADD IY,BC	FD09	15			?		0	*	"	"
ADD IY,DE	FD19	15			?		0	*	"	"
ADD IY,IY	FD29	15			?		0	*	"	"
ADD IY,SP	FD39	15			?		0	*	"	"
AND (HL)	A6	7	*	*	1	*	0	0	indirect	71
AND (IX+op)	DDA6op	19	*	*	1	*	0	0	indexé	"
AND (IY+op)	FDA6op	19	*	*	1	*	0	0	"	"
AND A	A7	4	*	*	1	*	0	0	registre	"
AND B	A0	4	*	*	1	*	0	0	"	"
AND C	A1	4	*	*	1	*	0	0	"	"
AND D	A2	4	*	*	1	*	0	0	"	"
AND E	A3	4	*	*	1	*	0	0	"	"
AND H	A4	4	*	*	1	*	0	0	"	"
AND L	A5	4	*	*	1	*	0	0	"	"
AND op	E6op	7	*	*	1	*	0	0	immédiat	"
BIT 0,(HL)	CB46	12	?	*	1	?	0		indirect	74
BIT 0, (IX+op)	DDCBop46	20	?	*	1	?	0		indexé	"
BIT 0, (IY+op)	FDCBop46	20	?	*	1	?	0		"	"
BIT 0,A	CB47	8	?	*	1	?	0		registre	"
BIT 0,B	CB40	8	?	*	1	?	0		"	"
BIT 0,C	CB41	8	?	*	1	?	0		"	"
BIT 0,D	CB42	8	?	*	1	?	0		"	"

Mnémonique	Code objet (hexa)	Cycles	S	Z	H	P/V	N	C	Adressage	Page
BIT 0,E	CB43	8	?	*	1	?	0		registre	74
BIT 0,H	CB44	8	?	*	1	?	0		"	"
BIT 0,L	CB45	8	?	*	1	?	0		"	"
BIT 1, (HL)	CB4E	12	?	*	1	?	0		indirect	"
BIT 1, (IX+op)	DDCBop4E	20	?	*	1	?	0		indexé	"
BIT 1, (IY+op)	FDCBop4E	20	?	*	1	?	0		"	"
BIT 1,A	CB4F	8	?	*	1	?	0		registre	"
BIT 1,B	CB48	8	?	*	1	?	0		"	"
BIT 1,C	CB49	8	?	*	1	?	0		"	"
BIT 1,D	CB4A	8	?	*	1	?	0		"	"
BIT 1,E	CB4B	8	?	*	1	?	0		"	"
BIT 1,H	CB4C	8	?	*	1	?	0		"	"
BIT 1,L	CB4D	8	?	*	1	?	0		"	"
BIT 2, (HL)	CB56	12	?	*	1	?	0		indirect	"
BIT 2, (IX+op)	DDCBop56	20	?	*	1	?	0		indexé	"
BIT 2, (IY+op)	FDCBop56	20	?	*	1	?	0		"	"
BIT 2,A	CB57	8	?	*	1	?	0		registre	"
BIT 2,B	CB50	8	?	*	1	?	0		"	"
BIT 2,C	CB51	8	?	*	1	?	0		"	"
BIT 2,D	CB52	8	?	*	1	?	0		"	"
BIT 2,E	CB53	8	?	*	1	?	0		"	"
BIT 2,H	CB54	8	?	*	1	?	0		"	"
BIT 2,L	CB55	8	?	*	1	?	0		"	"
BIT 3, (HL)	CB5E	12	?	*	1	?	0		indirect	"
BIT 3, (IX+op)	DDCBop5E	20	?	*	1	?	0		indexé	"
BIT 3, (IY+op)	FDCBop5E	20	?	*	1	?	0		"	"
BIT 3,A	CB5F	8	?	*	1	?	0		registre	"
BIT 3,B	CB58	8	?	*	1	?	0		"	"
BIT 3,C	CB59	8	?	*	1	?	0		"	"
BIT 3,D	CB5A	8	?	*	1	?	0		"	"
BIT 3,E	CB5B	8	?	*	1	?	0		"	"
BIT 3,H	CB5C	8	?	*	1	?	0		"	"
BIT 3,L	CB5D	8	?	*	1	?	0		"	"
BIT 4, (HL)	CB66	12	?	*	1	?	0		indirect	"
BIT 4, (IX+op)	DDCBop66	20	?	*	1	?	0		indexé	"
BIT 4, (IY+op)	FDCBop66	20	?	*	1	?	0		"	"
BIT 4,A	CB67	8	?	*	1	?	0		registre	"
BIT 4,B	CB60	8	?	*	1	?	0		"	"
BIT 4,C	CB61	8	?	*	1	?	0		"	"
BIT 4,D	CB62	8	?	*	1	?	0		"	"
BIT 4,E	CB63	8	?	*	1	?	0		"	"
BIT 4,H	CB64	8	?	*	1	?	0		"	"
BIT 4,L	CB65	8	?	*	1	?	0		"	"
BIT 5, (HL)	CB6E	12	?	*	1	?	0		indirect	"

Mnémonique	Code objet (hexa)	Cycles	S	Z	H	P/V	N	C	Adressage	Page
BIT 5, (IX+op)	DDCBop6E	20	?	*	1	?	0		indexé	74
BIT 5, (IY+op)	FDCBop6E	20	?	*	1	?	0		"	"
BIT 5,A	CB6F	8	?	*	1	?	0		registre	"
BIT 5,B	CB68	8	?	*	1	?	0		"	"
BIT 5,C	CB69	8	?	*	1	?	0		"	"
BIT 5,D	CB6A	8	?	*	1	?	0		"	"
BIT 5,E	CB6B	8	?	*	1	?	0		"	"
BIT 5,H	CB6C	8	?	*	1	?	0		"	"
BIT 5,L	CB6D	8	?	*	1	?	0		"	"
BIT 6, (HL)	CB76	12	?	*	1	?	0		indirect	"
BIT 6, (IX+op)	DDCBop76	20	?	*	1	?	0		indexé	"
BIT 6, (IY+op)	FDCBop76	20	?	*	1	?	0		"	"
BIT 6,A	CB77	8	?	*	1	?	0		registre	"
BIT 6,B	CB70	8	?	*	1	?	0		"	"
BIT 6,C	CB71	8	?	*	1	?	0		"	"
BIT 6,D	CB72	8	?	*	1	?	0			"
BIT 6,E	CB73	8	?	*	1	?	0			"
BIT 6,H	CB74	8	?	*	1	?	0			"
BIT 6,L	CB75	8	?	*	1	?	0			"
BIT 7, (HL)	CB7E	12	?	*	1	?	0		indirect	"
BIT 7, (IX+op)	DDCBop7E	20	?	*	1	?	0		indexé	"
BIT 7, (IY+op)	FDCBop7E	20	?	*	1	?	0		"	"
BIT 7,A	CB7F	8	?	*	1	?	0		registre	"
BIT 7,B	CB78	8	?	*	1	?	0		"	"
BIT 7,C	CB79	8	?	*	1	?	0		"	"
BIT 7,D	CB7A	8	?	*	1	?	0		"	"
BIT 7,E	CB7B	8	?	*	1	?	0		"	"
BIT 7,H	CB7C	8	?	*	1	?	0		"	"
BIT 7,L	CB7D	8	?	*	1	?	0		"	"
CALL C,ad	DCad	17 10							immédiat	96
CALL M,ad	FCad	17 10							"	"
CALL NC,ad	D4ad	17 10							"	"
CALL NZ,ad	C4ad	17 10							"	"
CALL P,ad	F4ad	17 10							"	"
CALL PE,ad	ECad	17 10							"	"
CALL PO,ad	E4ad	17 10							"	"
CALL Z,ad	CCad	17 10							"	"
CALL ad	CDad	17							"	95
CCF	3F	4			?		0	*	implicite	74
CP (HL)	BE	7	*	*	*	*	1	*	indirect	87
CP (IX+op)	DDBEop	19	*	*	*	*	1	*	indexé	"
CP (IY+op)	FDBEop	19	*	*	*	*	1	*	"	"
CP A	BF	4	*	*	*	*	1	*	registre	"
CP B	B8	4	*	*	*	*	1	*	"	"

Mnémonique	Code objet (hexa)	Cycles	S	Z	H	P/V	N	C	Adressage	Page
			Registre d'état							
CP C	B9	4	*	*	*	*	1	*	registre	87
CP D	BA	4	*	*	*	*	1	*	"	"
CP E	BB	4	*	*	*	*	1	*	"	"
CP H	BC	4	*	*	*	*	1	*	"	"
CP L	BD	4	*	*	*	*	1	*	"	"
CP op	FEop	7	*	*	*	*	1	*	immédiat	"
CPD	EDA9	16	*	*	*	*	1		implicite	88
CPDR	EDB9	16 21	*	*	*	*	1		"	89
CPI	EDA1	16	*	*	*	*	1		"	90
CPIR	EDB1	16 21	*	*	*	*	1		"	"
CPL	2F	4			1		1		"	69
DAA	27	4	*	*	*	*		*	"	70
DEC (HL)	35	11	*	*	*	*	1	*	indirect	68
DEC (IX+op)	DD35op	23	*	*	*	*	1	*	indexé	"
DEC (IY+op)	FD35op	23	*	*	*	*	1	*	"	"
DEC A	3D	4	*	*	*	*	1	*	registre	"
DEC B	05	4	*	*	*	*	1	*	"	"
DEC C	0D	4	*	*	*	*	1	*	"	"
DEC D	15	4	*	*	*	*	1	*	"	"
DEC E	1D	4	*	*	*	*	1	*	"	"
DEC H	25	4	*	*	*	*	1	*	"	"
DEC L	2D	4	*	*	*	*	1	*	"	"
DEC BC	OB	6							"	"
DEC DE	1B	6							"	"
DEC HL	2B	6							"	"
DEC SP	3B	6							"	"
DEC IX	DD2B	10							"	"
DEC IY	FD2B	10							"	"
DI	F3	4							implicite	105
DJNZ op	10op	8 13							relatif	93
EI	FB	4							implicite	105
EX (SP),HL	E3	19							indirect	57
EX (SP),IX	DDE3	23							"	"
EX (SP),IY	FDE3	23							"	"
EX AF,AF'	08	4							implicite	58
EX DE,HL	EB	4							"	"
EXX	D9	4							"	"
HALT	76	4							"	106
IM0	ED46	4							"	104
IM1	ED56	4							"	"
IM2	ED5E	4							"	"
IN A,(C)	ED78	12	*	*	*	*	0		indirect	99
IN B,(C)	ED40	12	*	*	*	*	0		"	"
IN C,(C)	ED48	12	*	*	*	*	0		"	"

Mnémonique	Code objet (hexa)	Cycles	S	Z	H	P V	N	C	Adressage	Page
IN D,(C)	ED50	12	*	*	*	*	0		indirect	99
IN E,(C;	ED58	12	*	*	*	*	0		"	"
IN H,(C)	ED60	12	*	*	*	*	0		"	"
IN L,(C)	ED68	12	*	*	*	*	0		"	"
IN A,(op)	DBop	11							"	100
INC (HL)	34	11	*	*	*	*	0		indirect	66
INC (IX+op)	DD34op	23	*	*	*	*	0		indexé	"
INC (IY+op)	FD34op	23	*	*	*	*	0		"	"
INC A	3C	4	*	*	*	*	0		registre	"
INC B	04	4	*	*	*	*	0		"	"
INC C	0C	4	*	*	*	*	0		"	"
INC D	14	4	*	*	*	*	0		"	"
INC E	1C	4	*	*	*	*	0		"	"
INC H	24	4	*	*	*	*	0		"	"
INC L	2C	4	*	*	*	*	0		"	"
INC BC	03	6							"	"
INC DE	13	6							"	"
INC HL	23	6							"	"
INC SP	33	6							"	"
INC IX	DD23	10							"	67
INC IY	FD23	10							"	"
IND	EDAA	16	?	*	?	?	1		implicite	100
INDR	EDBA	16/21	?	1	?	?	1		"	102
INI	EDA2	16	?	*	?	?	1		"	101
INIR	EDB2	16/21	?	1	?	?	1		"	102
JP ad	C3ad	10							immédiat	91
JP (HL)	E9	4							indirect	92
JP (IX)	DDE9	8							"	"
JP (IY)	FDE9	8							"	"
JP C,ad	DAad	10							immédiat	91
JP M,ad	FAad	10							"	"
JP NC,ad	D2ad	10							"	"
JP NZ,ad	C2ad	10							"	"
JP P,ad	F2ad	10							"	"
JP PE,ad	EAad	10							"	"
JP PO,ad	E2ad	10							"	"
JP Z,ad	CAad	12/7							"	"
JR C,op	38op	12/7							relatif	93
JR NC,op	30op	12/7							"	"
JR NZ,op	20op	12/7							"	"
JR Z,op	28op	12/7							"	"
JR op	18op	12							"	"
LD (BC),A	02	7							indirect	48
LD (DE),A	12	7							"	"

Mnémonique	Code objet (hexa)	Cycles	S	Z	H	P/V	N	C	Adressage	Page
LD (HL),A	77	7							indirect	48
LD (HL),B	70	7							"	"
LD (HL),C	71	7							"	"
LD (HL),D	72	7							"	"
LD (HL),E	73	7							"	"
LD (HL),H	74	7							"	"
LD (HL),L	75	7							"	"
LD (HL),op	36op	10							imm./ind.	"
LD (IX+op),A	DD77op	19							indexé	"
LD (IX+op),B	DD70op	19							"	"
LD (IX+op),C	DD71op	19							"	"
LD (IX+op),D	DD72op	19							"	"
LD (IX+op),E	DD73op	19							"	"
LD (IX+op),H	DD74op	19							"	"
LD (IX+op),L	DD75op	19							"	"
LD (IX+op),op'	DD36opop'	19							ind./imm.	"
LD (IY+op),A	FD77op	19							indexé	"
LD (IY+op),B	FD70op	19							"	"
LD (IY+op),C	FD71op	19							"	"
LD (IY+op),D	FD72op	19							"	"
LD (IY+op),E	FD73op	19							"	"
LD (IY+op),H	FD74op	19							"	"
LD (IY+op),L	FD75op	19							"	"
LD (IY+op),op'	FD36opop'	19							ind./imm.	"
LD (ad),A	32ad	13							direct	49
LD (ad),BC	ED43ad	20							"	"
LD (ad),DE	ED53ad	20							"	"
LD (ad),HL	22ad	16							"	"
LD (ad),IX	DD22ad	20							"	"
LD (ad),IY	FD22ad	20							"	"
LD (ad),SP	ED73ad	20							"	"
LD A, (BC)	0A	7							indirect	48
LD A, (DE)	1A	7							"	"
LD A, (HL)	7E	7							"	"
LD A, (IX+op)	DD7Eop	19							indexé	"
LD A, (IY+op)	FD7Eop	19							"	"
LD A, (ad)	3Aad	13							direct	"
LD A,A	7F	4							registre	"
LD A,B	78	4							"	"
LD A,C	79	4							"	"
LD A,D	7A	4							"	"
LD A,E	7B	4							"	"
LD A,H	7C	4							"	"
LD A,L	7D	4							"	"

Mnémonique	Code objet (hexa)	Cycles	Registre d'état						Adressage	Page
			S	Z	H	P/V	N	C		
LD A,op	3Eop	7							immédiat	48
LD A,I	ED57	9	*	*	0	*	0		implicite	52
LD A,R	ED5F	9	*	*	0	*	0		"	"
LD B, (HL)	46	7							indirect	48
LD B, (IX+op)	DD46op	19							indexé	"
LD B, (IY+op)	FD46op	19							"	"
LD B,A	47	4							registre	"
LD B,B	40	4							"	"
LD B,C	41	4							"	"
LD B,D	42	4							"	"
LD B,E	43	4							"	"
LD B,H	44	4							"	"
LD B,L	45	4							"	"
LD B,op	06op	7							immédiat	"
LD C,(HL)	4E	7							indirect	"
LD C, (IX+op)	DD4Eop	19							indexé	"
LD C, (IY+op)	FD4Eop	19							"	"
LD C,A	4F	4							registre	"
LD C,B	48	4							"	"
LD C,C	49	4							"	"
LD C,D	4A	4							"	"
LD C,E	4B	4							"	"
LD C,H	4C	4							"	"
LD C,L	4D	4							"	"
LD C,op	0Eop	7							immédiat	"
LD D, (HL)	56	7							indirect	"
LD D, (IX+op)	DD56op	19							indexé	"
LD D, (IY+op)	FD56op	19							"	"
LD D,A	57	4							registre	"
LD D,B	50	4							"	"
LD D,C	51	4							"	"
LD D,D	52	4							"	"
LD D,E	53	4							"	"
LD D,H	54	4							"	"
LD D,L	55	4							"	"
LD D,op	16op	7							immédiat	"
LD E, (HL)	5E	7							indirect	"
LD E, (IX+op)	DD5Eop	19							indexé	"
LD E, (IY+op)	FD5Eop	19							"	"
LD E,A	5F	4							registre	"
LD E,B	58	4							"	"
LD E,C	59	4							"	"
LD E,D	5A	4							"	"
LD E,E	5B	4							"	"

Mnémonique	Code objet (hexa)	Cycles	S	Z	H	P/V	N	C	Adressage	Page
LD E,H	5C	4							registre	48
LD E,L	5D	4							"	"
LD E,op	1Eop	7							immédiat	"
LD H, (HL)	66	7							indirect	"
LD H, (IX+op)	DD66op	19							indexé	"
LD H, (IY+op)	FD66op	19							"	"
LD H,A	67	4							registre	"
LD H,B	60	4							"	"
LD H,C	61	4							"	"
LD H,D	62	4							"	"
LD H,E	63	4							"	"
LD H,H	64	4							"	"
LD H,L	65	4							"	"
LD H,op	26op	7							immédiat	"
LD L, (HL)	6E	7							indirect	"
LD L, (IX+op)	DD6Eop	19							indexé	"
LD L, (IY+op)	FD6Eop	19							"	"
LD L,A	6F	4							registre	"
LD L,B	68	4							"	"
LD L,C	69	4							"	"
LD L,D	6A	4							"	"
LD L,E	6B	4							"	"
LD L,H	6C	4							"	"
LD L,L	6D	4							"	"
LD L,op	2Eop	7							immédiat	"
LD I,A	ED47	9							implicite	52
LD R,A	ED4F	9							"	"
LD BC,(ad)	ED4Bad	20							direct	49
LD BC,ad	01ad	10							immédiat	"
LD DE, (ad)	ED5Bad	20							direct	"
LD DE,ad	11ad	10							immédiat	"
LD HL, (ad)	2Aad	16							direct	"
LD HL,ad	21ad	10							immédiat	"
LD IX, (ad)	DD2Aad	20							direct	"
LD IX,ad	DD21ad	14							immédiat	"
LD IY,(ad)	FD2Aad	20							direct	"
LD IY,ad	FD21ad	14							immédiat	"
LD SP,(ad)	ED7Bad	20							direct	"
LD SP,HL	F9	6							implicite	53
LD SP,IX	DDF9	10							"	"
LD SP,IY	FDF9	10							"	"
LD SP,ad	31ad	10							immédiat	49
LDD	EDA8	16			0	*	0		indirect	54
LDDR	EDB8	16 21			0	0	0		"	55

Mnémonique	Code objet (hexa)	Cycles	S	Z	H	P/V	N	C	Adressage	Page
LDI	EDA0	16			0	*	0		indirect	55
LDIR	EDB0	16/21			0	0	0		"	56
NEG	ED44	8	*	*	*	*	1	*	implicite	70
NOP	00	4							"	105
OR (HL)	B6	7	*	*	0	*	0	0	indirect	72
OR (IX+op)	DDB6op	19	*	*	0	*	0	0	indexé	"
OR (IY+op)	FDB6op	19	*	*	0	*	0	0	"	"
OR A	B7	4	*	*	0	*	0	0	registre	"
OR B	B0	4	*	*	0	*	0	0	"	"
OR C	B1	4	*	*	0	*	0	0	"	"
OR D	B2	4	*	*	0	*	0	0	"	"
OR E	B3	4	*	*	0	*	0	0	"	"
OR H	B4	4	*	*	0	*	0	0	"	"
OR L	B5	4	*	*	0	*	0	0	"	"
OR op	F6op	7	*	*	0	*	0	0	immédiat	"
OTDR	EDBB	16/21	?	1	?	?	1		indirect	104
OTIR	EDB3	16/21	?	1	?	?	1		"	"
OUT (C),A	ED79	12							"	102
OUT (C),B	ED41	12							"	"
OUT (C),C	ED49	12							"	"
OUT (C),D	ED51	12							"	"
OUT (C),E	ED59	12							"	"
OUT (C),H	ED61	12							"	"
OUT (C),L	ED69	12							"	"
OUT (op),A	D3op	11							"	103
OUTD	EDAB	16	?	*	?	?	1		"	"
OUTI	EDA3	16	?	*	?	?	1		"	104
POP AF	F1	10							"	94
POP BC	C1	10							"	"
POP DE	D1	10							"	"
POP HL	E1	10							"	"
POP IX	DDE1	14							"	"
POP IY	FDE1	14							"	"
PUSH AF	F5	11							"	"
PUSH BC	C5	11							"	"
PUSH DE	D5	11							"	"
PUSH HL	E5	11							"	"
PUSH IX	DDE5	15							"	"
PUSH IY	FDE5	15							"	"
RES 0,(HL)	CB86	15	?	*	1	?	0		indirect	75
RES 0,(IX+op)	DDCBop86	23	?	*	1	?	0		indexé	"
RES 0,(IY+op)	FDCBop86	23	?	*	1	?	0		"	"
RES 0,A	CB87	8	?	*	1	?	0		registre	"
RES 0,B	CB80	8	?	*	1	?	0		"	"

Mnémonique	Code objet (hexa)	Cycles	S	Z	H	P V	N	C	Adressage	Page
RES 0.C	CB81	8	?	*	1	?	0		registre	75
RES 0.D	CB82	8	?	*	1	?	0		"	"
RES 0.E	CB83	8	?	*	1	?	0		"	"
RES 0.H	CB84	8	?	*	1	?	0		"	"
RES 0.L	CB85	8	?	*	1	?	0		"	"
RES 1.(HL)	CB8E	15	?	*	1	?	0		indirect	"
RES 1.(IX + op)	DDCBop8E	23	?	*	1	?	0		indexé	"
RES 1.(IY + op)	FDCBop8E	23	?	*	1	?	0		"	"
RES 1.A	CB8F	8	?	*	1	?	0		registre	"
RES 1.B	CB88	8	?	*	1	?	0		"	"
RES 1.C	CB89	8	?	*	1	?	0		"	"
RES 1.D	CB8A	8	?	*	1	?	0		"	"
RES 1.E	CB8B	8	?	*	1	?	0		"	"
RES 1.H	CB8C	8	?	*	1	?	0		"	"
RES 1.L	CB8D	8	?	*	1	?	0		"	"
RES 2. (HL)	CB96	15	?	*	1	?	0		indirect	"
RES 2. (IX + op)	DDCBop96	23	?	*	1	?	0		indexé	"
RES 2. (IY + op)	FDCBop96	23	?	*	1	?	0		"	"
RES 2.A	CB97	8	?	*	1	?	0		registre	"
RES 2.B	CB90	8	?	*	1	?	0		"	"
RES 2.C	CB91	8	?	*	1	?	0		"	"
RES 2.D	CB92	8	?	*	1	?	0		"	"
RES 2.E	CB93	8	?	*	1	?	0		"	"
RES 2.H	CB94	8	?	*	1	?	0		"	"
RES 2.L	CB95	8	?	*	1	?	0		"	"
RES 3. (HL)	CB9E	15	?	*	1	?	0		indirect	"
RES 3. (IX + op)	DDCBop9E	23	?	*	1	?	0		indexé	"
RES 3. (IY + op)	FDCBop9E	23	?	*	1	?	0		"	"
RES 3.A	CB9F	8	?	*	1	?	0		registre	"
RES 3.B	CB98	8	?	*	1	?	0		"	"
RES 3.C	CB99	8	?	*	1	?	0		"	"
RES 3.D	CB9A	8	?	*	1	?	0		"	"
RES 3.E	CB9B	8	?	*	1	?	0		"	"
RES 3.H	CB9C	8	?	*	1	?	0		"	"
RES 3.L	CB9D	8	?	*	1	?	0		"	"
RES 4. (HL)	CBA6	15	?	*	1	?	0		indirect	"
RES 4. (IX + op)	DDCBopA6	23	?	*	1	?	0		indexé	"
RES 4. (IY + op)	FDCBopA6	23	?	*	1	?	0		"	"
RES 4.A	CBA7	8	?	*	1	?	0		registre	"
RES 4.B	CBA0	8	?	*	1	?	0		"	"
RES 4.C	CBA1	8	?	*	1	?	0		"	"
RES 4.D	CBA2	8	?	*	1	?	0		"	"
RES 4.E	CBA3	8	?	*	1	?	0		"	"
RES 4.H	CBA4	8	?	*	1	?	0		"	"

Mnémonique	Code objet (hexa)	Cycles	S	Z	H	P V	N	C	Adressage	Page
RES 4,L	CBA5	8	?	*	1	?	0		registre	75
RES 5, (HL)	CBAE	15	?	*	1	?	0		indirect	"
RES 5, (IX+op)	DDCBopAE	23	?	*	1	?	0		indexé	"
RES 5, (IY+op)	FDCBopAE	23	?	*	1	?	0		"	"
RES 5,A	CBAF	8	?	*	1	?	0		registre	"
RES 5,B	CBA8	8	?	*	1	?	0		"	"
RES 5,C	CBA9	8	?	*	1	?	0		"	"
RES 5,D	CBAA	8	?	*	1	?	0		"	"
RES 5,E	CBAB	8	?	*	1	?	0		"	"
RES 5,H	CBAC	8	?	*	1	?	0		"	"
RES 5,L	CBAD	8	?	*	1	?	0		"	"
RES 6, (HL)	CBB6	15	?	*	1	?	0		indirect	"
RES 6, (IX+op)	DDCBopB6	23	?	*	1	?	0		indexé	"
RES 6, (IY+op)	FDCBopB6	23	?	*	1	?	0		"	"
RES 6,A	CBB7	8	?	*	1	?	0		registre	"
RES 6,B	CBB0	8	?	*	1	?	0		"	"
RES 6,C	CBB1	8	?	*	1	?	0		"	"
RES 6,D	CBB2	8	?	*	1	?	0		"	"
RES 6,E	CBB3	8	?	*	1	?	0		"	"
RES 6,H	CBB4	8	?	*	1	?	0		"	"
RES 6,L	CBB5	8	?	*	1	?	0		"	"
RES 7, (HL)	CBBE	15	?	*	1	?	0		indirect	"
RES 7, (IX+op)	DDCBopBE	23	?	*	1	?	0		indexé	"
RES 7, (IY+op)	FDCBopBE	23	?	*	1	?	0		"	"
RES 7,A	CBBF	8	?	*	1	?	0		registre	"
RES 7,B	CBB8	8	?	*	1	?	0		"	"
RES 7,C	CBB9	8	?	*	1	?	0		"	"
RES 7,D	CBBA	8	?	*	1	?	0		"	"
RES 7,E	CBBB	8	?	*	1	?	0		"	"
RES 7,H	CBBC	8	?	*	1	?	0		"	"
RES 7,L	CBBD	8	?	*	1	?	0		"	"
RET	C9	10							indirect	97
RET C	D8	5 11							"	"
RET M	F8	5 11							"	"
RET NC	D0	5 11							"	"
RET NZ	C0	5 11							"	"
RET P	F0	5 11							"	"
RET PE	E8	5 11							"	"
RET PO	E0	5 11							"	"
RET Z	C8	5 11							"	"
RETI	ED4D	14							"	98
RETN	ED45	14							"	"
RL (HL)	CB16	15	*	*	0	*	0	*	indirect	76
RL (IX+op)	DDCBop16	23	*	*	0	*	0	*	indexé	"

Mnémonique	Code objet (hexa)	Cycles	Registre d'état S	Z	H	P V	N	C	Adressage	Page
RL (IY+op)	FDCBop16	23	*	*	0	*	0	*	indexé	76
RL A	CB17	8	*	*	0	*	0	*	registre	"
RL B	CB10	8	*	*	0	*	0	*	"	"
RL C	CB11	8	*	*	0	*	0	*	"	"
RL D	CB12	8	*	*	0	*	0	*	"	"
RL E	CB13	8	*	*	0	*	0	*	"	"
RL H	CB14	8	*	*	0	*	0	*	"	"
RL L	CB15	8	*	*	0	*	0	*	"	"
RLA	17	4			0		0	*	implicite	77
RLC (HL)	CB06	15	*	*	0	*	0	*	indirect	80
RLC (IX+op)	DDCBop06	23	*	*	0	*	0	*	indexé	"
RLC (IY+op)	FDCBop06	23	*	*	0	*	0	*	"	"
RLC A	CB07	8	*	*	0	*	0	*	registre	"
RLC B	CB00	8	*	*	0	*	0	*	"	"
RLC C	CB01	8	*	*	0	*	0	*	"	"
RLC D	CB02	8	*	*	0	*	0	*	"	"
RLC E	CB03	8	*	*	0	*	0	*	"	"
RLC H	CB04	8	*	*	0	*	0	*	"	"
RLC L	CB05	8	*	*	0	*	0	*	"	"
RLCA	07	4			0		0	*	implicite	84
RLD	ED6F	18	*	*	0	*	0		"	86
RR (HL)	CB1E	15	*	*	0	*	0	*	indirect	78
RR (IX+op)	DDCBop1E	23	*	*	0	*	0	*	indexé	"
RR (IY+op)	FDCBop1E	23	*	*	0	*	0	*	"	"
RR A	CB1F	8	*	*	0	*	0	*	registre	"
RR B	CB18	8	*	*	0	*	0	*	"	"
RR C	CB19	8	*	*	0	*	0	*	"	"
RR D	CB1A	8	*	*	0	*	0	*	"	"
RR E	CB1B	8	*	*	0	*	0	*	"	"
RR H	CB1C	8	*	*	0	*	0	*	"	"
RR L	CB1D	8	*	*	0	*	0	*	"	"
RRA	1F	4			0		0	*	implicite	79
RRC (HL)	CB0E	15	*	*	0	*	0	*	indirect	82
RRC (IX+op)	DDCBop0E	23	*	*	0	*	0	*	indexé	"
RRC (IY+op)	FDCBop0E	23	*	*	0	*	0	*	"	"
RRC A	CB0F	8	*	*	0	*	0	*	registre	"
RRC B	CB08	8	*	*	0	*	0	*	"	"
RRC C	CB09	8	*	*	0	*	0	*	"	"
RRC D	CB0A	8	*	*	0	*	0	*	"	"
RRC E	CB0B	8	*	*	0	*	0	*	"	"
RRC H	CB0C	8	*	*	0	*	0	*	"	"
RRC L	CB0D	8	*	*	0	*	0	*	"	"
RRCA	0F	4			0	*	0	*	implicite	"
RRD	ED67	18	*	*	0	*	0		indirect	86

Mnémonique	Code objet (hexa)	Cycles	S	Z	H	P/V	N	C	Adressage	Page
RST 00h	C7	11							indirect	98
RST 08h	CF	11							"	"
RST 10h	D7	11							"	"
RST 18h	DF	11							"	"
RST 20h	E7	11							"	"
RST 28h	EF	11							"	"
RST 30h	F7	11							"	"
RTS 38h	FF	11							"	"
SBC A,op	DEop	7	*	*	*	*	1	*	immédiat	63
SBC A, (HL)	9E	7	*	*	*	*	1	*	indirect	"
SBC A, (IX+op)	DD9E	19	*	*	*	*	1	*	indexé	"
SBC A, (IY+op)	FD9E	19	*	*	*	*	1	*	"	"
SBC A,A	9F	4	*	*	*	*	1	*	registre	"
SBC A,B	98	4	*	*	*	*	1	*	"	"
SBC A,C	99	4	*	*	*	*	1	*	"	"
SBC A,D	9A	4	*	*	*	*	1	*	"	"
SBC A,E	9B	4	*	*	*	*	1	*	"	"
SBC A,H	9C	4	*	*	*	*	1	*	"	"
SBC A,L	9D	4	*	*	*	*	1	*	"	"
SBC HL,BC	ED42	15	*	*	?	*	1	*	"	64
SBC HL,DE	ED52	15	*	*	?	*	1	*	"	"
SBC HL,HL	ED62	15	*	*	?	*	1	*	"	"
SBC HL,SP	ED72	15	*	*	?	*	1	*	"	"
SCF	37	4			0		0	1	implicite	73
SET 0, (HL)	CBC6	15	?	*	1	?	0		indirect	75
SET 0, (IX+op)	DDCBopC6	23	?	*	1	?	0		indexé	"
SET 0, (IY+op)	FDCBopC6	23	?	*	1	?	0		"	"
SET 0,A	CBC7	8	?	*	1	?	0		registre	"
SET 0,B	CBC0	8	?	*	1	?	0		"	"
SET 0,C	CBC1	8	?	*	1	?	0		"	"
SET 0,D	CBC2	8	?	*	1	?	0		"	"
SET 0,E	CBC3	8	?	*	1	?	0		"	"
SET 0,H	CBC4	8	?	*	1	?	0		"	"
SET 0,L	CBC5	8	?	*	1	?	0		"	"
SET 1, (HL)	CBCE	15	?	*	1	?	0		indirect	"
SET 1, (IX+op)	DDCBopCE	23	?	*	1	?	0		indexé	"
SET 1, (IY+op)	FDCBopCE	23	?	*	1	?	0		"	"
SET 1,A	CBCF	8	?	*	1	?	0		registre	"
SET 1,B	CBC8	8	?	*	1	?	0		"	"
SET 1,C	CBC9	8	?	*	1	?	0		"	"
SET 1,D	CBCA	8	?	*	1	?	0		"	"
SET 1,E	CBCB	8	?	*	1	?	0		"	"
SET 1,H	CBCC	8	?	*	1	?	0		"	"
SET 1,L	CBCD	8	?	*	1	?	0		"	"

Mnémonique	Code objet (hexa)	Cycles	Registre d'état S	Z	H	P/V	N	C	Adressage	Page
SET 2, (HL)	CBD6	15	?	*	1	?	0		indirect	75
SET 2, (IX+op)	DDCBopD6	23	?	*	1	?	0		indexé	"
SET 2, (IY+op)	FDCBopD6	23	?	*	1	?	0		"	"
SET 2,A	CBD7	8	?	*	1	?	0		registre	"
SET 2,B	CBD0	8	?	*	1	?	0		"	"
SET 2,C	CBD1	8	?	*	1	?	0		"	"
SET 2,D	CBD2	8	?	*	1	?	0		"	"
SET 2,E	CBD3	8	?	*	1	?	0		"	"
SET 2,H	CBD4	8	?	*	1	?	0		"	"
SET 2,L	CBD5	8	?	*	1	?	0		"	"
SET 3, (HL)	CBDE	15	?	*	1	?	0		indirect	"
SET 3, (IX+op)	DDCBopDE	23	?	*	1	?	0		indexé	"
SET 3, (IY+op)	FDCBopDE	23	?	*	1	?	0		"	"
SET 3,A	CBDF	8	?	*	1	?	0		registre	"
SET 3,B	CBD8	8	?	*	1	?	0		"	"
SET 3,C	CBD9	8	?	*	1	?	0		"	"
SET 3,D	CBDA	8	?	*	1	?	0		"	"
SET 3,E	CBDB	8	?	*	1	?	0		"	"
SET 3,H	CBDC	8	?	*	1	?	0		"	"
SET 3,L	CBDD	8	?	*	*	?	0		"	"
SET 4, (HL)	CBE6	15	?	*	1	?	0		indirect	"
SET 4, (IX+op)	DDCBopE6	23	?	*	1	?	0		indexé	"
SET 4, (IY+op)	FDCBopE6	23	?	*	1	?	0		"	"
SET 4,A	CBE7	8	?	*	1	?	0		registre	"
SET 4,B	CBE0	8	?	*	1	?	0		"	"
SET 4,C	CBE1	8	?	*	1	?	0		"	"
SET 4,D	CBE2	8	?	*	1	?	0		"	"
SET 4,E	CBE3	8	?	*	1	?	0		"	"
SET 4,H	CBE4	8	?	*	1	?	0		"	"
SET 4,L	CBE5	8	?	*	1	?	0		"	"
SET 5, (HL)	CBEE	15	?	*	1	?	0		indirect	"
SET 5, (IX+op)	DDCBopEE	23	?	*	1	?	0		indexé	"
SET 5, (IY+op)	FDCBopEE	23	?	*	1	?	0		"	"
SET 5,A	CBEF	8	?	*	1	?	0		registre	"
SET 5,B	CBE8	8	?	*	1	?	0		"	"
SET 5,C	CBE9	8	?	*	1	?	0		"	"
SET 5,D	CBEA	8	?	*	1	?	0		"	"
SET 5,E	CBEB	8	?	*	1	?	0		"	"
SET 5,H	CBEC	8	?	*	1	?	0		"	"
SET 5,L	CBED	8	?	*	1	?	0		"	"
SET 6, (HL)	CBF6	15	?	*	1	?	0		indirect	"
SET 6, (IX+op)	DDCBopF6	23	?	*	1	?	0		indexé	"
SET 6, (IY+op)	FDCBopF6	23	?	*	1	?	0		"	"
SET 6,A	CBF7	8	?	*	1	?	0		registre	"

Mnémonique	Code objet (hexa)	Cycles	S	Z	H	P/V	N	C	Adressage	Page
SET 6,B	CBF0	8	?	*	1	?	0		registre	75
SET 6,C	CBF1	8	?	*	1	?	0		"	"
SET 6,D	CBF2	8	?	*	1	?	0		"	"
SET 6,E	CBF3	8	?	*	1	?	0		"	"
SET 6,H	CBF4	8	?	*	1	?	0		"	"
SET 6,L	CBF5	8	?	*	1	?	0		"	"
SET 7, (HL)	CBFE	15	?	*	1	?	0		indirect	"
SET 7, (IX+op)	DDCBopFE	23	?	*	1	?	0		indexé	"
SET 7, (IY+op)	FDCBopFE	23	?	*	1	?	0		"	"
SET 7,A	CBFF	8	?	*	1	?	0		registre	"
SET 7,B	CBF8	8	?	*	1	?	0		"	"
SET 7,C	CBF9	8	?	*	1	?	0		"	"
SET 7,D	CBFA	8	?	*	1	?	0		"	"
SET 7,E	CBFB	8	?	*	1	?	0		"	"
SET 7,H	CBFC	8	?	*	1	?	0		"	"
SET 7,L	CBFD	8	?	*	1	?	0		"	"
SLA (HL)	CB26	15	*	*	0	*	0	*	indirect	83
SLA (IX+op)	DDCBop26	23	*	*	0	*	0	*	indexé	"
SLA (IY+op)	FDCBop26	23	*	*	0	*	0	*	"	"
SLA A	CB27	8	*	*	0	*	0	*	registre	"
SLA B	CB20	8	*	*	0	*	0	*	"	"
SLA C	CB21	8	*	*	0	*	0	*	"	"
SLA D	CB22	8	*	*	0	*	0	*	"	"
SLA E	CB23	8	*	*	0	*	0	*	"	"
SLA H	CB24	8	*	*	0	*	0	*	"	"
SLA L	CB25	8	*	*	0	*	0	*	"	"
SRA (HL)	CB2E	15	*	*	0	*	0	*	indirect	84
SRA (IX+op)	DDCBop2E	23	*	*	0	*	0	*	indexé	"
SRA (IY+op)	FDCBop2E	23	*	*	0	*	0	*	"	"
SRA A	CB2F	8	*	*	0	*	0	*	registre	"
SRA B	CB28	8	*	*	0	*	0	*	"	"
SRA C	CB29	8	*	*	0	*	0	*	"	"
SRA D	CB2A	8	*	*	0	*	0	*	"	"
SRA E	CB2B	8	*	*	0	*	0	*	"	"
SRA H	CB2C	8	*	*	0	*	0	*	"	"
SRA L	CB2D	8	*	*	0	*	0	*	"	"
SRL (HL)	CB3E	15	*	*	0	*	0	*	indirect	85
SRL (IX+op)	DDCBop3E	23	*	*	0	*	0	*	indexé	"
SRL (IY+op)	FDCBop3E	23	*	*	0	*	0	*	"	"
SRL A	CB3F	8	*	*	0	*	0	*	registre	"
SRL B	CB38	8	*	*	0	*	0	*	"	"
SRL C	CB39	8	*	*	0	*	0	*	"	"
SRL D	CB3A	8	*	*	0	*	0	*	"	"
SRL E	CB3B	8	*	*	0	*	0	*	"	"

Mnémonique	Code objet (hexa)	Cycles	S	Z	H	P.V	N	C	Adressage	Page
SRL H	CB3C	8	*	*	0	*	0	*	registre	85
SRL L	CB3D	8	*	*	0	*	0	*	"	"
SUB (HL)	96	7	*	*	*	*	1	*	indirect	65
SUB (IX+op)	DD96op	19	*	*	*	*	1	*	indexé	"
SUB (IY+op)	FD96op	19	*	*	*	*	1	*	"	"
SUB A	97	4	*	*	*	*	1	*	registre	"
SUB B	90	4	*	*	*	*	1	*	"	"
SUB C	91	4	*	*	*	*	1	*	"	"
SUB D	92	4	*	*	*	*	1	*	"	"
SUB E	93	4	*	*	*	*	1	*	"	"
SUB H	94	4	*	*	*	*	1	*	"	"
SUB L	95	4	*	*	*	*	1	*	"	"
SUB op	D6op	7	*	*	*	*	1	*	immédiat	"
XOR (HL)	AE	7	*	*	0	*	0	0	indirect	73
XOR (IX+op)	DDAEop	19	*	*	0	*	0	0	indexé	"
XOR (IY+op)	FDAEop	19	*	*	0	*	0	0	"	"
XOR A	AF	4	*	*	0	*	0	0	registre	"
XOR B	A8	4	*	*	0	*	0	0	"	"
XOR C	A9	4	*	*	0	*	0	0	"	"
XOR D	AA	4	*	*	0	*	0	0	"	"
XOR E	AB	4	*	*	0	*	0	0	"	"
XOR H	AC	4	*	*	0	*	0	0	"	"
XOR L	AD	4	*	*	0	*	0	0	"	"
XOR op	EEop	7	*	*	0	*	0	0	"	"

Index

Achevé d'imprimer en avril 1985
sur les presses de l'imprimerie Laballery et Cie
58500 Clamecy
Dépôt légal : avril 1985

N° d'impression : 503005
N° d'édition : 86595-239-1
ISBN : 2-86595-239-8

Votre avis nous intéresse

- Pour nous permettre de faire de meilleurs livres, adressez-nous vos critiques sur le présent livre.
- Si vous souhaitez des eclaircissements techniques, écrivez-nous, nous adresserons votre demande a l'auteur qui ne manquera pas de vous répondre directement.

- Ce livre vous donne-t-il toute satisfaction?

- Y a-t-il un aspect du problème que vous auriez aimé voir abordé?

Comment avez-vous eu connaissance de ce livre?

- [ ] publicite
- [ ] catalogue
- [ ] boutique micro
- [ ] autres
- [ ] cadeau
- [ ] librairie
- [ ] exposition

Avez-vous déjà acquis des livres PSI?

lesquels?

qu'en pensez-vous?

Nom | Prenom | Age

Adresse

Profession

Centre d'interêt



ASSEMBLEUR ET PÉRIPHÉRIQUES DES MSX

"Assembleur et périphériques des MSX" s'adresse à vous qui avez une bonne pratique du BASIC et souhaitez programmer votre MSX en langage machine.

Vous vous initierez, dans un premier temps, aux principes de base de l'assembleur du Z 80 pour appliquer vos connaissances, dans un deuxième temps, aux particularités des MSX : VDP, générateur de sons, BIOS, ROM et région de communication, etc.

Le jeu d'instructions du Z 80 vous permettra d'obtenir rapidité d'exécution et économie de mémoire, tandis que des routines et adresses utiles vous apprendront à utiliser à fond les périphériques des MSX.

De nombreux exemples et exercices, dont les solutions sont données en annexe, vous aideront à maîtriser votre système MSX.



ÉDITIONS DU P.S.I.
BP 86 – 77402 LAGNY S/MARNE CEDEX – FRANCE

ISBN : 2 86595-239 8 **110 F.F.**

PHOTO : L. MAUMOURY